WebSphere MQ

Application Messaging Interface

SC34-6065-00

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix D, “Notices” on page 625.

First edition (March 2002)

This edition applies to IBM® WebSphere® MQ Application Messaging Interface Version 1.2.2, and to any subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 5. C object interface reference 83

Part 5. The Java interface 373

Chapter 14. Using the Application Messaging Interface in Java 377

Chapter 15. Java interface overview 391

Chapter 16. Java interface reference 405

Figures

Tables

About this book

This book describes how to use the WebSphere MQ Application Messaging Interface. The Application Messaging Interface provides a simple interface that application programmers can use without needing to understand all the details of the WebSphere MQ Message Queue Interface.

Who this book is for

This book is for anyone who wants to use the Application Messaging Interface to send and receive WebSphere MQ messages, including publish/subscribe and point-to-point applications.

What you need to know to understand this book

- Knowledge of the C, COBOL, C++, or Java™ programming language is assumed.
- You don’t need previous experience of WebSphere MQ to use the Application Messaging Interface (AMI). You can use the examples and sample programs provided to find out how to send and receive messages. However, to understand all the functions of the AMI you need to have some knowledge of the WebSphere MQ Message Queue Interface (MQI). This is described in the *WebSphere MQ Application Programming Guide* and the *WebSphere MQ Application Programming Reference* book.
- You will need to read the following:
  - *MQSeries Publish/Subscribe User’s Guide* if you are going to use the AMI with MQSeries® Publish/Subscribe.
  - *MQSeries Integrator Version 1.1 Application Development Guide* if you are going to use the AMI with MQSeries Integrator Version 1.1.
  - *WebSphere MQ Integrator Version 2.1 Programming Guide* if you are going to use the AMI with WebSphere MQ Integrator Version 2.1.
- If you are a systems administrator responsible for setting up an installation of the AMI, you need to be experienced in using the MQI.

Structure of this book

This book contains the following parts:

- Part 1, “Introduction” on page 1 gives an overview of the Application Messaging Interface.
- Part 2, “The C interface” on page 9 describes how to use the AMI in C programs. If you are new to WebSphere MQ, gain some experience with the high-level interface first. It provides most of the functionality you need when writing applications. Then move on to the object interface if you need extra functionality.
- Part 3, “The C++ interface” on page 161 describes how to use the AMI in C++ programs.
- Part 4, “The COBOL interface” on page 241 describes how to write AMI programs using the COBOL high-level and object interfaces.
- Part 5, “The Java interface” on page 373 describes how to use the AMI in Java programs.

- Part 6, "Setting up an AMI installation" on page 441 is for systems administrators who are setting up an Application Messaging Interface installation.

Conventions used in this book

This book uses the following type styles:

`Format` The name of a parameter in a WebSphere MQ call, a field in a WebSphere MQ structure, or an attribute of a WebSphere MQ object

amInitialize
The name of an AMI function or method

AMB_TRUE
The name of an AMI constant

`AmString getName();`
The syntax of AMI functions and methods, and example code

The term Windows® refers to Microsoft® Windows 98, Windows NT®, Windows Me, and Windows 2000, unless explicitly stated otherwise.

| The term UNIX® refers to AIX®, HP-UX, and Solaris, unless explicitly stated
| otherwise.

Part 1. Introduction

Chapter 1. Introduction

The WebSphere MQ products enable programs to communicate with one another across a network of dissimilar components - processors, operating systems, subsystems, and communication protocols - using a consistent application programming interface, the WebSphere MQ *Message Queue Interface* (MQI). The *Application Messaging Interface* (AMI) provides a simple interface that application programmers can use without needing to understand all the functions available in the MQI. The functions that are required in a particular installation are defined by a system administrator, using *services* and *policies*.

Main components of the AMI

There are three main components in the AMI:

- The message, which defines *what* is sent from one program to another
- The service, which defines *where* the message is sent
- The policy, which defines *how* the message is sent

To send a message using the AMI, an application has to specify the message data, together with the service and policy to use. You can use the default services and policies provided by the system, or create your own. Optionally, you can store your definitions of services and policies in a *repository*.

Sending and receiving messages

You can use the AMI to send and receive messages in a number of different ways:

- Send and forget (datagram), where no reply is needed
- Distribution list, where a message is sent to multiple destinations
- Request/response, where a sending application needs a response to the request message
- Publish/subscribe, where a broker manages the distribution of messages

Interoperability

The AMI is interoperable with other WebSphere MQ interfaces. Using the AMI, you can exchange messages with one or more of the following:

- Another application that is using the AMI
- Any application that is using the MQI
- A message broker (such as WebSphere MQ Publish/Subscribe or WebSphere MQ Integrator)

Programming languages

The Application Messaging Interface is available in the C, COBOL, C++, and Java programming languages. In C and COBOL, there are two interfaces: a high-level interface that is procedural in style, and a lower level object-style interface. The high-level interface contains the functionality needed by the majority of applications. You can mix the two interfaces as required.

In C++ and Java, a single object interface is provided.

Description of the AMI

In the Application Messaging Interface, messages, services and policies define what is sent, where it is sent, and how it is sent.

Messages

Information is passed between communicating applications using messages, with WebSphere MQ providing the transport. Messages consist of:

- The message attributes: information that identifies the message and its properties. The AMI uses the attributes, together with information in the policy, to interpret and construct WebSphere MQ headers and message descriptors.
- The message data: the application data carried in the message. The AMI does not act upon this data.

Some examples of message attributes are:

`MessageID`	An identifier for the message. It is usually unique, and typically it is generated by the message transport (WebSphere MQ).
`CorrelID`	A correlation identifier that can be used as a key, for example to correlate a response message to a request message. The AMI normally sets this in a response message by copying the `MessageID` from the request message.
`Format`	The structure of the message.
`Topic`	Indicates the content of the message for publish/subscribe applications.

These attributes are properties of an AMI message object. Where it is appropriate, an application can set them before sending a message, or access them after receiving a message. The message data can be contained in the message object, or passed as a separate parameter.

In a WebSphere MQ application, the message attributes are set up explicitly using the Message Queue Interface (MQI), so the application programmer needs to understand their purpose. With the AMI, they are contained in the message object, or defined in a policy that is set up by the system administrator, so the programmer is not concerned with these details.

Services

A service represents a destination that applications send messages to or receive messages from. In WebSphere MQ such a destination is called a *message queue*, and a queue resides in a *queue manager*. Programs can use the MQI to put messages on queues, and get messages from them. Because there are many parameters that are associated with queues, and because of the way queues are set up and managed, this interface is complex. When using the AMI, these parameters are defined in a service that the systems administrator sets up, so the complexity is hidden from the application programmer.

For further information about queues and queue managers, refer to the *WebSphere MQ Application Programming Guide*.

Point-to-point and publish/subscribe

In a *point-to-point* application, the sending application knows the destination of the message. Point-to-point applications can be send and forget (or datagram), where a reply to the message is not required, or request/response, where the request message specifies the destination for the response message. Applications using distribution lists to send a message to multiple destinations are usually of the send and forget type.

In the case of *publish/subscribe* applications, the providers of information are decoupled from the consumers of that information. The provider of the information is called a *publisher*. Publishers supply information about a subject by sending it to a broker. The subject is identified by a *topic*, such as "Stock" or "Weather". A publisher can publish information on more than one topic, and many publishers can publish information on a particular topic.

The consumer of the information is called a *subscriber*. A subscriber decides what information it is interested in, and subscribes to the relevant topics by sending a message to the broker. When information is published on one of those topics, the publish/subscribe broker sends it to the subscriber (and any others who have registered an interest in that topic). Each subscriber is sent information about those topics it has subscribed to.

There can be many brokers in a publish/subscribe system, and they communicate with each other to exchange subscription requests and publications. A publication is propagated to another broker if a subscription to that topic exists on the other broker. So a subscriber that subscribes to one broker will receive publications (on a chosen topic) that are published at another broker.

The AMI provides functions to send and receive messages using the publish/subscribe model. For further details, see the *MQSeries Publish/Subscribe User's Guide*.

Types of service

Different types of service are defined to specify the mapping from the AMI to real resources in the messaging network.

- Senders and receivers establish one-way communication pipes for sending and receiving messages.
- A distribution list contains a list of senders to which messages can be sent.
- A publisher contains a sender that is used to publish messages to a publish/subscribe broker.
- A subscriber contains a sender, used to subscribe to a publish/subscribe broker, and a receiver, used to receive publications from the broker.

The AMI provides default services that are used unless otherwise specified by the application program. You can define your own service when calling a function, or use a customized service stored in a *repository* (these are set up by a systems administrator). You do not have to have a repository. Many of the options used by the services are contained in a policy (see the next section).

The AMI has functions to open and close services explicitly, but they can also be opened and closed implicitly by other functions.

Policies

A policy controls how the AMI functions operate. Policies control such items as:

- The attributes of the message, for example, the priority
- Options for send and receive operations, for example, whether an operation is part of a unit of work
- Publish/subscribe options, for example, whether a publication is retained

- Added value functions that can be invoked as part of the call, such as retry

The AMI provides default policies. Alternatively, a systems administrator can define customized policies and store them in a repository. An application program selects a policy by specifying it as a parameter on calls.

You could choose to use a different policy on each call, and specify in the policy only those parameters that are relevant to the particular call. You could then have policies shared between applications, such as a “Transactional_Persistent_Put” policy. Another approach is to have policies that specify all the parameters for all the calls made in a particular application, such as a “Payroll_Client” policy. Both approaches are valid with the AMI, but a single policy for each application will simplify management of policies.

The AMI will automatically retry when temporary errors are encountered on sending a message, if requested by the policy. (Examples of temporary errors are queue full, queue disabled, and queue in use.)

Application Messaging Interface model

Figure 1 on page 7 shows the components of the Application Messaging Interface.

Application programs communicate directly with AMI objects using the object interface in C, COBOL, C++ and Java. In addition to the object-style interface, there is a procedural-style high-level interface available in C and COBOL. This contains the functionality needed by the majority of applications; it can be supplemented with object interface functions as needed.

Sender, receiver, distribution list, publisher, and subscriber objects are all services. Senders and receivers connect directly to the message transport layer (WebSphere MQ). Distribution list and publisher objects contain senders; subscriber objects contain a sender and a receiver.

Message, service, and policy objects are created and managed by a session object, which provides the scope for a unit of work. The session object contains a connection object that is not visible to the application. The combination of connection, sender, and receiver objects provides the transport for the message. Other objects, such as helper classes, are provided in C++ and Java.

Attributes for message, service, and policy objects can be taken from the system defaults, or from administrator-provided definitions that have been stored in the repository.

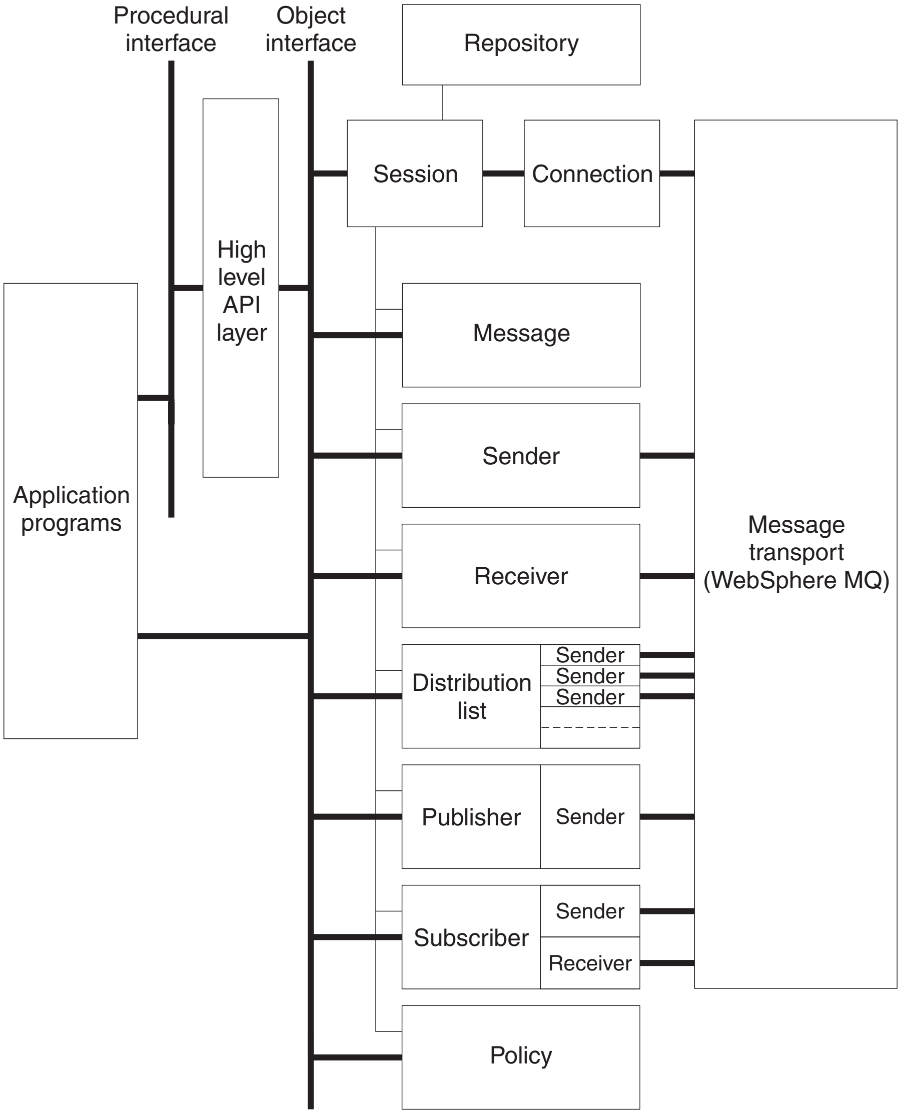

Figure 1. Basic AMI model

Further information

The syntax of the AMI differs according to the programming language, so the implementation for each language is described in a separate part of this book:

- Part 2, “The C interface” on page 9
- Part 3, “The C++ interface” on page 161
- Part 4, “The COBOL interface” on page 241
- Part 5, “The Java interface” on page 373

In Part 6, “Setting up an AMI installation” on page 441, you can find out how to:

- Install the Application Messaging Interface
- Run the sample programs
- Determine the cause of problems
- Set up services and policies

Further information

The Application Messaging Interface for C, C++, and Java runs on the following operating systems or environments: AIX, HP-UX, iSeries™, Solaris, Microsoft Windows 98, Windows NT, Windows Me, and Windows 2000.

The Application Messaging Interface for C and COBOL runs on the z/OS™ operating system.

Part 2. The C interface

Chapter 2. Using the Application Messaging Interface in C

The Application Messaging Interface (AMI) in the C programming language has two interfaces:

1. A high-level procedural interface that provides the function needed by most users.
2. A lower-level, object-style interface, that provides additional function for experienced WebSphere MQ users.

This chapter describes the following:
- "Structure of the AMI"
- "Writing applications in C" on page 16
- "Building C applications" on page 31

Structure of the AMI

Although the high-level interface is procedural in style, the underlying structure of the AMI is object based. (The term *object* is used here in the object-oriented programming sense, not in the sense of WebSphere MQ 'objects' such as channels and queues.) The objects that are made available to the application are:

Session	Contains the AMI session.
Connection	Manages the connection to the underlying message transport. This object is not visible to the application. The AMI creates this object implicitly and it sits beneath the session object.
Message	Contains the message data, message ID, correlation ID, and options that are used when sending or receiving a message (most of which come from the policy definition).
Sender	This is a service that represents a destination (such as a WebSphere MQ queue) to which messages are sent.
Receiver	This is a service that represents a source from which messages are received.
Distribution list	Contains a list of sender services to provide a list of destinations.
Publisher	Contains a sender service where the destination is a publish/subscribe broker.
Subscriber	Contains a sender service (to send subscribe and unsubscribe messages to a publish/subscribe broker) and a receiver service (to receive publications from the broker).
Policy	Defines how the message should be handled, including items such as priority, persistence, and whether it is included in a unit of work.
Policy handler	Defines policy handler libraries, that is, collections of functions that extend the function of the AMI. These objects are not visible to the application. The AMI creates these objects implicitly, based on the contents of the repository.

When the high-level functions are used, the objects are created automatically and (where applicable) populated with values from the repository. In some cases, it might be necessary to inspect these properties after a message has been sent (for example, the `MessageID`), or to change the value of one or more properties before sending the message (for example, the `Format`). To satisfy these requirements, the AMI for C has a lower-level object style interface in addition to the high-level procedural interface. This provides access to the objects listed earlier, with methods to *set* and *get* their properties. You can mix high-level and object-level functions in the same application.

All the objects have both a *handle* and a *name*. The names are used to access objects from the high-level interface. The handles are used to access them from the object interface. Multiple objects of the same type can be created with the same name, but are usable only from the object interface.

The high-level interface is described in Chapter 3, "The C high-level interface" on page 43. An overview of the object interface is given in Chapter 4, "C object interface overview" on page 69, with reference information in Chapter 5, "C object interface reference" on page 83.

Using the repository

You can run AMI applications with or without a repository. If you do not have a repository, you can use a system default object (see the next section), or create your own by specifying its name on a function call. It will be created using the appropriate system provided definition (see "System provided definitions" on page 492).

If you do have a repository, and you specify the name of an object on a function call that matches a name in the repository, the object will be created using the repository definition. (If no matching name is found in the repository, the system provided definition will be used.)

System default objects

Table 1. System default objects

Default object	Constant or handle (if applicable)
SYSTEM.DEFAULT.POLICY	AMSD_POL AMSD_POL_HANDLE
SYSTEM.DEFAULT.SYNCPOINT.POLICY	AMSD_SYNC_POINT_POL AMSD_SYNC_POINT_POL_HANDLE
SYSTEM.DEFAULT.SENDER	AMSD_SND
SYSTEM.DEFAULT.RESPONSE.SENDER	AMSD_RSP_SND AMSD_RSP_SND_HANDLE
SYSTEM.DEFAULT.RECEIVER	AMSD_RCV AMSD_RCV_HANDLE
SYSTEM.DEFAULT.PUBLISHER	AMSD_PUB AMSD_PUB_SND
SYSTEM.DEFAULT.SUBSCRIBER	AMSD_SUB AMSD_SUB_SND
SYSTEM.DEFAULT.SEND.MESSAGE	AMSD_SND_MSG AMSD_SND_MSG_HANDLE

Table 1. System default objects (continued)

Default object	Constant or handle (if applicable)
SYSTEM.DEFAULT.RECEIVE.MESSAGE	AMSD_RCV_MSG AMSD_RCV_MSG_HANDLE

A set of system default objects is created at session creation time. This removes the overhead of creating the objects from applications using these defaults. The system default objects are available for use from both the high-level and object interfaces in C. They are created using the system provided definitions (see "System provided definitions" on page 492).

The default objects can be specified explicitly using AMI constants, or used to provide defaults if a parameter is omitted (by specifying NULL, for example).

Constants representing synonyms for handles are also provided for these objects, for use from the object interface (see Appendix B, "Constants and structures" on page 561). Note that the first parameter on a call must be a real handle; you cannot use a synonym in this case (that is why handles are not provided for all the default objects).

Writing applications in C

This section gives a number of examples showing how to use the high-level interface of the AMI, with some extensions using the object interface. Equivalent operations to all high-level functions can be performed using combinations of object interface functions (see "High-level functions" on page 80).

Opening and closing a session

Before using the AMI, you must open a session. You can do this with the following high-level function (page 51):

Opening a session

```
hSession = amInitialize(name, myPolicy, &compCode, &reason);
```

The `name` is optional, and can be specified as NULL. `myPolicy` is the name of the policy to be used during initialization of the AMI. You can specify the policy name as NULL, in which case the SYSTEM.DEFAULT.POLICY object is used.

The function returns a *session handle*, which must be used by other calls in this session. Errors are returned using a completion code and reason code.

To close a session, you can use the following high-level function (page 66):

Closing a session

```
success = amTerminate(&hSession, myPolicy, &compCode, &reason);
```

This closes and deletes all objects that were created in the session. Note that a *pointer* to the session handle is passed. If the function is successful, it returns AMB_TRUE.

Sending messages

You can send a datagram (send and forget) message using the high-level **amSendMsg** function (page 62). In the simplest case, all you need to specify is the session handle returned by **amInitialize**, the message data, and the message length. Other parameters are set to NULL, so the default message, sender service, and policy objects are used.

Sending a message using all the defaults

```
success = amSendMsg(hSession, NULL, NULL, dataLen,
       pData, NULL, &compCode, &reason);
```

If you want to send the message using a different sender service, specify its name (such as `mySender`) as follows:

Sending a message using a specified sender service

```
success = amSendMsg(hSession, mySender, NULL, dataLen,
       pData, NULL, &compCode, &reason);
```

If you are not using the default policy, you can specify a policy name:

Sending a message using a specified policy

```
success = amSendMsg(hSession, NULL, myPolicy, dataLen,
       pData, NULL, &compCode, &reason);
```

The policy controls the behavior of the send function. For example, the policy can specify:

- The priority, persistence and expiry of the message
- If the send is part of a unit of work
- If the sender service should be implicitly opened and left open

To send a message to a distribution list, specify its name (such as `myDistList`) as the sender service:

Sending a message to a distribution list

```
success = amSendMsg(hSession, myDistList, NULL, dataLen,
       pData, NULL, &compCode, &reason);
```

Using the message object

Using the object interface gives you more functions when sending a message. For example, you can *get* or *set* individual attributes in the message object. To get an attribute after the message has been sent, you can specify a name for the message object that is being sent:

Specifying a message object

```
success = amSendMsg(hSession, NULL, NULL, dataLen,
       pData, mySendMsg, &compCode, &reason);
```

The AMI creates a message object of the name specified (`mySendMsg`), if one doesn’t already exist. (The sender name and policy name are specified as NULL, so in this example their defaults are used.) You can then use object interface functions to get the required attributes, such as the *`MessageID`*, from the message object:

Getting an attribute from a message object

```
hMsg = amSesGetMessageHandle(hSession, mySendMsg, &compCode, &reason);

success = amMsgGetMsgId(hMsg, BUFLEN, &MsgIdLen, pMsgId,
       &compCode, &reason);
```

The first call is needed to get the handle to the message object. The second call returns the message ID length, and the message ID itself (in a buffer of length BUFLEN).

To set an attribute such as the *Format* before the message is sent, you must first create a message object and set the format:

Setting an attribute in a message object

```
hMsg = amSesCreateMessage(hSession, mySendMsg, &compCode, &reason);

success = amMsgSetFormat(hMsg, AMLEN_NULL_TERM, pFormat,
            &compCode, &reason);
```

Then you can send the message as before, making sure to specify the same message object name (`mySendMsg`) in the **amSendMsg** call.

Look at "Message interface functions" on page 72 to find out what other attributes of the message object you can get and set.

After a message object has been used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see **amMsgReset** on page 109) and rebuild it each time.

Instead of sending the message data using the data buffer, it can be added to the message object. However, this is not recommended for large messages because of the overhead of copying the data into the message object before it is sent (and also extracting the data from the message object when it is received).

Sample programs

For more details, refer to the `amtshsnd.c` and `amtsosnd.c` sample programs (see "Sample programs for iSeries, UNIX, and Windows" on page 480).

Receiving messages

Use the **amReceiveMsg** high-level function (page 55) to receive a message to which no response is to be sent (such as a datagram). In the simplest case, all you need to specify are the session handle and a buffer for the message data. Other parameters are set to NULL, so the default message, receiver service, and policy objects are used.

Receiving a message using all the defaults

```
success = amReceiveMsg(hSession, NULL, NULL, NULL, BUFLEN,
       &dataLen, pData, NULL, &compCode, &reason);
```

If you want to receive the message using a different receiver service, specify its name (such as `myReceiver`) as follows:

Receiving a message using a specified receiver service

```
success = amReceiveMsg(hSession, myReceiver, NULL, NULL, BUFLEN,
       &dataLen, pData, NULL, &compCode, &reason);
```

If you are not using the default policy, you can specify a policy name:

Receiving a message using a specified policy

```
success = amReceiveMsg(hSession, NULL, myPolicy, NULL, BUFLEN,
      &dataLen, pData, NULL, &compCode, &reason);
```

The policy can specify, for example:
- The wait interval
- Whether the message is part of a unit of work
- Whether the message should be code page converted
- Whether all the members of a group must be there before any members can be read

Using the message object

To get the attributes of a message after receiving it, you can specify your own message object name, or use the system default (SYSTEM.DEFAULT.RECEIVE.MESSAGE). If a message object of that name does not exist it will be created. You can access the attributes (such as the *`Encoding`*) using the object interface functions:

Getting an attribute from a message object

```
success = amReceiveMsg(hSession, NULL, NULL, NULL, BUFLEN,
      &dataLen, pData, myRcvMsg, &compCode, &reason);

hMsg = amSessGetMessageHandle(hSession, myRcvMsg, &compCode, &reason);

success = amMsgGetEncoding(hMsg, &encoding, &compCode, &reason);
```

If a specific message is to be selectively received using its correlation identifier, a message object must first be created and its *`CorrelId`* property set to the required value (using the object interface). This message object is passed as the *selection message* on the **amReceiveMsg** call:

Using a selection message object

```
hMsg = amSesCreateMessage(hSession, mySelMsg, &compCode, &reason);

success = amMsgSetCorrelId(hMsg, correlIdLen, pCorrelId,
      &compCode, &reason);

success = amReceiveMsg(hSession, NULL, NULL, mySelMsg, BUFLEN,
      &dataLen, pData, NULL, &compCode, &reason);
```

Sample programs

For more details, refer to the `amtshrcv.c` and `amtsorcv.c` sample programs (see “Sample programs for iSeries, UNIX, and Windows” on page 480).

Request/response messaging

In the *request/response* style of messaging, a requester (or client) application sends a request message and expects to receive a message in response. The responder (or server) application receives the request message and produces the response message (or messages) which it returns to the requester application. The responder application uses information in the request message to determine how to send the response message to the requester.

In the following examples 'your' refers to the responding application (the server); 'my' refers to the requesting application (the client).

Request

Use the **amSendRequest** high-level function (page 63) to send a request message. This is similar to **amSendMsg**, but it includes the name of the service to which the response message is to be sent. In this example the sender service (`mySender`) is specified in addition to the receiver service (`myReceiver`). (A policy name and a send message name can be specified as well, as described in "Sending messages" on page 16.)

Sending a request message

```
success = amSendRequest(hSession, mySender, NULL, myReceiver,
       dataLen, pData, NULL, &compCode, &reason);
```

The **amReceiveRequest** high-level function (page 59) is used by the responding (or server) application to receive a request message. It is similar to **amReceiveMsg**, but it includes the name of the sender service that will be used for sending the response message. When the message is received, the sender service is updated with the information needed for sending the response to the required destination.

Receiving a request message

```
success = amReceiveRequest(hSession, yourReceiver, NULL, BUFLEN,
       &dataLen, pData, yourRcvMsg, yourSender,
       &compCode, &reason);
```

A policy name can be specified as well, as described in "Receiving messages" on page 18.

A receiver message name (`yourRcvMsg`) is specified so that the response message can refer to it. Note that, unlike **amReceiveMsg**, this function does not have a selection message.

Response

After the requested actions have been performed, the responding application sends the response message (or messages) with the **amSendResponse** function (page 64):

Sending a response message

```
success = amSendResponse(hSession, yourSender, NULL, yourRcvMsg,
       dataLen, pData, NULL, &compCode, &reason);
```

The sender service for the response message (`yourSender`) and the receiver message name (`yourRcvMsg`) are the same as those used with **amReceiveRequest**. This causes the *`CorrelId`* and *`MessageId`* to be set in the response message, as requested by the flags in the request message.

Finally, the requester (or client) application uses the **amReceiveMsg** function to receive the response message as described in "Receiving messages" on page 18. You might need to receive a specific response message (for example if three request messages have been sent, and you want to receive the response to the first request message first). In this case the sender message name from the **amSendRequest** function should be used as the selection message name in the **amReceiveMsg**.

Sample programs

For more details, refer to the `amtshclt.c`, `amtshsvr.c`, `amtsoclt.c`, and `amtsosvr.c` sample programs (see "Sample programs for iSeries, UNIX, and Windows" on page 480).

File transfer

You can perform file transfers using the **amSendFile** and **amReceiveFile** high-level functions, and the **amSndSendFile**, **amDstSendFile** and **amRcvReceiveFile** object-level functions. There are two broad applications of the file transfer calls: end-to-end file transfer using both send file and receive file calls, and generation of messages from a file using just a send file call. If the message supplied to the send file call has a format of AMFMT_STRING (the default), the file is treated as text. If the format is AMFMT_NONE, the file is treated as binary data and is not converted in any way.

Sending a file using the high-level amSendFile function

```
success = amSendFile(hSession, mySender, myPolicy, 0, 0, NULL,
       AMLEN_NULL_TERM, "myFilename", mySendMessage,
       &compCode, &reason);
```

Use a policy with the 'physical splitting' file transfer option, to ensure that the file can be reassembled at the receiving side during end-to-end file transfer. With this mode of file transfer, the AMI passes extra meta-data with the file to help ensure that the complete file is recovered and to allow the original filename to travel with the message.

When using physical splitting, the AMI might send a group of messages instead of one large message. This implies that, when sending files to or receiving files on platforms without native group support, AMI simulated groups must be used. See "Sending group messages" on page 27 for more information. Because errors might occur part way through sending or receiving a file, you must ensure that the transfer completed as expected. You can do this using the syncpoint policy option, and checking the reason and completion codes to be sure that the whole file was sent before committing the unit of work.

Receiving a file using the high-level amReceiveFile function

```
success = amReceiveFile(hSession, myReceiver, myPolicy, 0,
       mySelectionMessage, 0, NULL, 0, NULL, myReceiveMessage,
       &compCode, &reason);
```

If the message selected for the receive operation does not contain file information, it is returned to the application in the message object named on the call and a warning is returned with reason AMRC_NOT_A_FILE. If the file transfer fails part way through a message, that message is returned to the application and the current data pointer within the message shows how far it had been processed before the error occurred. Again, you can use the syncpoint policy option, checking the completion and reason codes to ensure the whole file was received correctly before committing the unit of work. If the file was sent from a different type of file system than it is received into, the AMI converts the file and returns a warning with reason AMRC_FILE_FORMAT_CONVERTED. This conversion allows transfer between z/OS datasets with different record types or sizes, and between z/OS datasets and the flat files used on other systems.

To generate a group of messages from a file (instead of transferring a file from one location to another), you can use the 'logical splitting' policy option. If the message object referenced by the send call has a format of AMFMT_STRING, the file is split into lines and each line is sent as a separate message. Any other format indicates that the file does not contain text. If the record length of a non-text file is known (as in the case of z/OS datasets) then each record is sent as a separate message. If the record length of a non-text file is not known then the whole file is considered to be a single record, and is sent in one message. No extra header information is added to the file data. The messages can then be processed in the same fashion as any other message in your queueing network.

On z/OS, if a new file is created by a file receive operation the AMI allocates an initial extent of 40 tracks and an extension size of 100 tracks. This allows a maximum capacity of approximately 69MB. To override this allocation:

- include a DD statement with appropriate parameters in your JCL
- specify the AMI function ’fileName’ parameter as ’DD:outfile’, where ’outfile’ is the JCL DD name
- specify File Disposition as ’Overwrite’ in the policy receive attributes

If the dataset full condition is reached before all the data is written, the AMI file receive function returns completion code AMCC_WARNING with reason code AMRC_FILE_TRUNCATED. This condition might result in lost data. To preserve data integrity, include the file receive function call within a logical unit of work, and take appropriate action (for example, backout the unit of work) if AMRC_FILE_TRUNCATED is received.

On z/OS, a file receive operation for a file sent from a non-z/OS platform normally completes with completion code AMCC_WARNING and reason code AMRC_FILE_FORMAT_CONVERTED. By default, the file is allocated the attributes `recfm=VB`, `lrecl=1028`. If this allocation is inappropriate, for instance because the record length is insufficient, you can override the allocation using a JCL DD statement, as described above.

Note that file transfer calls are not supported under CICS®. An error with reason code AMRC_FILE_TRANSFER_INVALID (144) is returned if any file transfer call is used in a CICS application running on z/OS.

Publish/subscribe messaging

With *publish/subscribe* messaging, *publisher* applications publish messages to *subscriber* applications using a *broker*. The messages published contain application data and one or more *topic* strings that describe the data. Subscribing applications register subscriptions informing the broker which topics they are interested in. When the broker receives a published message, it forwards the message to all subscribing applications for which a topic in the message matches a topic in the subscription.

Subscribing applications can exploit content-based publish/subscribe by passing a filter on subscribe and unsubscribe calls (see “Using WebSphere MQ Integrator Version 2” on page 477).

For more information, refer to the *WebSphere MQ Integrator Version 2 Programming Guide* or the *WebSphere MQ Publish/Subscribe User’s Guide*.

Publish

Use the **amPublish** high-level function (page 52) to publish a message. You need to specify the name of the publisher for the publish/subscribe broker. The topic relating to this publication and the publication data must also be specified:

Publishing a message

```
success = amPublish(hSession, myPublisher, NULL, myReceiver,
      strlen(topic), pTopic, dataLen, pData, myPubMsg,
      &compCode, &reason);
```

The name `myReceiver` identifies the receiver service to which the broker will send a response message. You can also specify a policy name to change the behavior of the function (as with the **amSend** functions).

You can specify the publication message name `myPubMsg` and set or get attributes of the message object (using the object interface functions). This might include adding another topic (using **amMsgAddTopic**) before invoking **amPublish**, if there are multiple topics associated with this publication.

Instead of sending the publication data using the data buffer, it can be added to the message object. Unlike the **amSend** functions, this gives no difference in performance with large messages. This is because, whichever method is used, the MQRFH header has to be added to the publication data before sending it (similarly the header has to be removed when the publication is received).

Subscribe

The **amSubscribe** high-level function (page 65) is used to subscribe to a publish/subscribe broker specified by the name of a subscriber service. The receiver to which publications will be sent is included within the definition of the subscriber. The name of a receiver service to which the broker can send a response message (`myReceiver`) is also specified.

Subscribing to a broker

```
success = amSubscribe(hSession, mySubscriber, NULL, myReceiver,
      strlen(topic), pTopic, 0L, NULL, mySubMsg,
      &compCode, &reason);
```

A subscription for a single topic can be passed by the `pTopic` parameter. You can subscribe to multiple topics by using the object interface **amMsgAddTopic** function to add topics to the subscription message object, before invoking **amSubscribe**.

If the policy specifies that the *`CorrelId`* is to be used as part of the identity for the subscribing application, it can be added to the subscription message object with the object interface **amMsgSetCorrelId** function, before invoking **amSubscribe**.

To remove a subscription, use the **amUnsubscribe** high-level function (page 67). To remove all subscriptions, you can specify a policy that has the ‘Deregister All Topics’ subscriber attribute.

To receive a publication from a broker, use the **amReceivePublication** function (page 57). For example:

Receiving a publication

```
success = amReceivePublication(hSession, mySubscriber, NULL, NULL,
       TOPICBUFLEN, BUFLEN, &topicCount, &topicLen, pFirstTopic,
       &dataLen, pData, myRcvMsg, &compCode, &reason);
```

You need to specify the name of the subscriber service used for the original subscription. You can also specify a policy name and a selection message name, as described in “Receiving messages” on page 18, but they are shown as NULL in this example.

If there are multiple topics associated with the publication, only the first one is returned by this function. So, if `topicCount` indicates that there are more topics, you have to access them from the `myRcvMsg` message object, using the object-level **amSesGetMessageHandle** and **amMsgGetTopic** functions.

Sample programs

For more details, refer to the `amtshpub.c`, `amtshsub.c`, `amtsopub.c`, and `amtsosub.c` sample programs (see “Sample programs for iSeries, UNIX, and Windows” on page 480).

Using name/value elements

Publish/subscribe brokers (such as WebSphere MQ Publish/Subscribe) respond to messages that contain name/value pairs to define the commands and options to be used. The **amPublish**, **amSubscribe**, **amUnsubscribe**, and **amReceivePublication** high-level functions provide these name/value pairs implicitly.

For less commonly used commands and options, the name/value pairs can be added to a message using an AMELEM structure, which is defined as follows:

```
typedef struct tagAMELEM {
  AMCHAR8  strucId;       /* Structure identifier     */
  AMLONG   version;       /* Structure version number */
  AMLONG   groupBuffLen;  /* Reserved, must be zero   */
  AMLONG   groupLen;      /* Reserved, must be zero   */
  AMSTR    pGroup;        /* Reserved, must be NULL   */
  AMLONG   nameBuffLen;   /* Name buffer length       */
  AMLONG   nameLen;       /* Name length in bytes     */
  AMSTR    pName;         /* Name                     */
  AMLONG   valueBuffLen;  /* Value buffer length      */
  AMLONG   valueLen;      /* Value length in bytes    */
  AMSTR    pValue;        /* Value                    */
  AMLONG   typeBuffLen;   /* Reserved, must be zero   */
  AMLONG   typeLen;       /* Reserved, must be zero   */
  AMSTR    pType;         /* Reserved, must be NULL   */
 } AMELEM;
```

See “Initial values for structures” on page 31 for advice on initialization of this structure.

Parameters

strucId	The AMELEM structure identifier (input). Its value must be AMELEM_STRUC_ID. The constant AMELEM_STRUC_ID_ARRAY is also defined; this has the same value as AMELEM_STRUC_ID but is an array of characters instead of a string.
version	The version number of the AMELEM structure (input). Its value must be AMELEM_VERSION_1.
groupBuffLen	Reserved, must be zero.
groupLen	Reserved, must be zero.
pGroup	Reserved, must be NULL.
nameBuffLen	The length of the name buffer (input). If the `nameBuffLen` parameter value is set to 0, the AMI returns the `nameLen` value but not the `pName` value. This is not an error.
nameLen	The length of the name in bytes (input or output). A value of AMLEN_NULL_TERM denotes a null-terminated string of unspecified length.
pName	The name buffer (input or output).
valueBuffLen	The length of the value buffer (input). If `valueBuffLen` is set to zero, the AMI returns the `valueLen` value but not the `pValue` value. This is not an error.
valueLen	The value length in bytes (input or output). A value of AMLEN_NULL_TERM denotes a null-terminated string of unspecified length.
pValue	The value buffer (input or output).
typeBuffLen	Reserved, must be zero.
typeLen	Reserved, must be zero.
pType	Reserved, must be NULL.

Example

As an example, to send a message containing a ‘Request Update’ command, initialize the AMELEM structure and then set the following values:

pName	AMPS_COMMAND
pValue	AMPS_REQUEST_UPDATE

Having set the values, create a message object (`mySndMsg`) and add the element to it:

Using name/value elements

```
hMsg = amSessCreateMessage(hSession, mySndMsg, &compCode, &reason);

success = amMsgAddElement(hMsg, pElem, 0L, &compCode, &reason);
```

You must then send the message, using **amSendMsg**, to the sender service specified for the publish/subscribe broker.

If you need to use streams with WebSphere MQ Publish/Subscribe, you must add the appropriate stream name/value element explicitly to the message object. Helper macros (such as **AmMsgAddStreamName**) are provided to simplify this and other tasks.

The message element functions can, in fact, be used to add any element to a message before issuing a publish/subscribe request. Such elements (including topics, which are specialized elements) supplement or override those added implicitly by the request, as appropriate to the individual element type.

The use of name/value elements is not restricted to publish/subscribe applications. They can be used in other applications as well.

Error handling

Each AMI C function returns a completion code reflecting the success or failure (OK, warning, or error) of the request. Information indicating the reason for a warning or error is returned in a reason code. Both completion and reason codes are optional.

Also, each function returns an AMBOOL value or an AMI object handle. For functions that return an AMBOOL value, this value is set to AMB_TRUE if the function completes successfully or with a warning, and to AMB_FALSE if an error occurs.

The ‘get last error’ functions (such as **amSesGetLastError**) always reflect the last most severe error detected by an object. These functions can be used to return the completion and reason codes associated with this error. Once the error has been handled, call the ‘clear error codes’ functions (for example, **amMsgClearErrorCodes**) to clear the error information.

All C high-level functions record last error information in the session object. This information can be accessed using the session’s ‘get last error’ call, **amSesGetLastError** (you need the session handle returned by **amInitialize** as the first parameter of this call).

Transaction support

Messages sent and received by the AMI can, optionally, be part of a transactional unit of work. A message is included in a unit of work based on the setting of the syncpoint attribute specified in the policy used on the call. The scope of the unit of work is the session handle and only one unit of work may be active at any time.

The API calls used to control the transaction depend on the type of transaction being used.

- WebSphere MQ messages are the only resource

  A transaction is started by the first message sent or received under syncpoint control, as specified in the policy specified for the send or receive. Multiple messages can be included in the same unit of work. The transaction is committed or backed out using an **amCommit** or **amBackout** high-level interface call (or the **amSesCommit** or **amSesRollback** object-level calls).
- Using WebSphere MQ as an XA transaction coordinator

  The transaction must be started explicitly using the **amSesBegin** call before the first recoverable resource (such as a relational database) is changed. The transaction is committed or backed out using an **amCommit** or **amBackout** high-level interface call (or the **amSesCommit** or **amSesRollback** object-level calls).

WebSphere MQ cannot be used as an XA transaction coordinator on z/OS.

- Using an external transaction coordinator

  The transaction is controlled using the API calls of an external transaction coordinator (such as CICS, Encina® or Tuxedo). The AMI calls are not used but the syncpoint attribute must still be specified in the policy used on the call.

Sending group messages

The AMI allows a sequence of related messages to be included in, and sent as, a message group. Group context information is sent with each message to allow the message sequence to be preserved and made available to a receiving application. To include messages in a group, the group status information of the first and subsequent messages in the group must be set as follows:

```
AMGRP_FIRST_MSG_IN_GROUP for the first message
AMGRP_MIDDLE_MSG_IN_GROUP for all messages other than first and last
AMGRP_LAST_MSG_IN_GROUP for the last message
```

The message status is set using **amMsgSetGroupStatus**.

To enable a queue for native group message support on WebSphere MQ for z/OS, V5.3, the queue attribute INDXTYPE must be set to GROUPID. Queues that are not indexed by GROUPID, or queues on earlier versions of the product on this platform (such as MQSeries for OS/390®, V5.2), can make use of simulated group messages. On other platforms, native group support is always enabled.

Simulated group messages

If native group support is not available, you can use the AMI to send and receive simulated group messages. This is done by selecting the ‘Simulated Group Support’ option in the repository service point definitions of the sender and receiver services. You must select this option for both sender and receiver, even if one of them has native group support.

Applications can send and receive group messages in the same way as for native group support, but certain restrictions apply when using simulated group support. These are as follows:

- Applications cannot set or use the correlation identifier.
- A message that is not part of a group is sent as a group of one message (that is, its group flags are set to specify it is the only message in a group).
- When receiving a message, the ‘Open shared’ receive policy option must be enabled (the default).
- Any non-simulated group messages that are on the same underlying queue will be ignored by the receive request.

Applications for z/OS subsystems

Here is some advice on writing AMI applications for the IMS™, CICS, batch, and RRS-batch subsystems on z/OS.

IMS applications

In an IMS application, you establish a syncpoint by using IMS calls such as GU (get unique) to the IOPCB and CHKP (checkpoint). To back out changes since the previous checkpoint, you can use the IMS ROLB (rollback) call. For more information, see the following manuals:

- *IMS/ESA® Application Programming: Transaction Manager*
- *IMS/ESA Application Programming: Design Guide*

If other recoverable resources are also involved in the unit of work, the queue manager (in conjunction with the IMS syncpoint manager) participates in a two-phase commit protocol; otherwise, the queue manager performs a single-phase commit process.

All AMI sessions are marked as expired at a syncpoint or rollback (except in a batch-orientated BMP). This is because a different user could initiate the next unit of work and WebSphere MQ security checking is performed when an AMI session or service is opened, not when an AMI object is accessed.

Any subsequent use of a session that has been marked expired (or any object created using that session), will return AMRC_SESSION_EXPIRED. It is the application's responsibility to ensure that all AMI sessions marked as expired are actually deleted.

It is recommended that applications explicitly end all AMI sessions (using **amSesDelete** or **amTerminate**) before syncpoint, to ensure that any AMI reason codes are correctly reported to the application, and to help ensure that all AMI sessions are deleted.

If an IMS application closes or deletes an AMI session, no implicit syncpoint is taken. If the application closes down normally, any open services are closed and an implicit commit occurs. If the application closes down abnormally, any open services are closed and an implicit backout occurs.

CICS applications

In a CICS application, you establish a syncpoint by using CICS calls such as EXEC CICS SYNCPOINT. To back out changes to the previous syncpoint you can use the EXEC CICS SYNCPOINT ROLLBACK call. For more information, see the *CICS Application Programming Reference* manual.

If other recoverable resources are also involved in the unit of work, the queue manager (in conjunction with the CICS syncpoint manager) participates in a two-phase commit protocol; otherwise, the queue manager performs a single-phase commit process.

If a CICS application closes or deletes an AMI session, no implicit syncpoint is taken. If the application closes down normally, any open services are closed and an implicit commit occurs. If the application closes down abnormally, any open services are closed and an implicit backout occurs. Note that file transfer calls are not supported under CICS. If used in a CICS application on z/OS, they return the reason code: AMRC_FILE_TRANSFER_INVALID (144).

If the AMI detects an internal processing error from which no recovery is possible, CICS applications will create a CICS transaction dump, with identifier 'MAMT'. See "First failure symptom report (z/OS)" on page 532 for more details.

Batch applications

In a batch application, you establish a syncpoint by using AMI calls such as **amCommit** or **amSesCommit**. To back out changes to the previous syncpoint you can use the **amBackout** or **amSesRollback** calls.

Note: If you need to commit or back out updates to resources managed by different resource managers, such as WebSphere MQ and DB2®, within a single unit of work, you should use RRS. For further information, see "RRS-batch applications".

If a batch application closes or deletes an AMI session, an implicit syncpoint is taken. If the application closes down normally, without first closing or deleting an AMI session, an implicit syncpoint occurs. If the application closes down abnormally, an implicit backout occurs.

RRS-batch applications

Transaction management and recoverable resource services (RRS) is an z/OS facility that provides two-phase syncpoint support across participating resource managers. An application can update recoverable resources managed by various z/OS resource managers such as WebSphere MQ and DB2 and then commit or back out these changes as a single unit of work.

In a RRS-batch application, you establish a syncpoint by using RRS calls such as SRRCMIT. To back out changes to the previous syncpoint you can use the SRRBACK call. For more information, see the *MVS™ Callable Services for High Level Languages* manual.

RRS availability: If RRS is not active on your z/OS system, any AMI call that resolves to a WebSphere MQ call will return one of the following AMI reason codes:

AMI reason code	Reason code 2
AMRC_TRANSPORT_ERROR	MQRC_ENVIRONMENT_ERROR
AMRC_BACKOUT_INVALID	NONE
AMRC_COMMIT_INVALID	NONE

If an RRS application closes or deletes an AMI session, no implicit syncpoint is taken. If the application closes down normally, any open services are closed and an implicit commit occurs. If the application closes down abnormally, any open services are closed and an implicit backout occurs.

Other considerations

You should consider the following when writing your applications:
- Multithreading
- Using WebSphere MQ with the AMI
- Field limits
- Using the AMI OAMAS subset

Multithreading

If you are using multithreading with the AMI, a session normally remains locked for the duration of a single AMI call. If you use receive with wait, the session remains locked for the duration of the wait, which might be unlimited (that is, until the wait time is exceeded or a message arrives on the queue). If you want another thread to run while a thread is waiting for a message, it must use a separate session.

AMI handles and object references can be used on a different thread from that on which they were first created for operations that do not involve an access to the underlying (WebSphere MQ) message transport. Functions such as initialize, terminate, open, close, send, receive, publish, subscribe, unsubscribe, and receive publication will access the underlying transport restricting these to the thread on which the session was first opened (for example, using **amInitialize** or **amSesOpen**). An attempt to issue these on a different thread will cause an error to be returned by WebSphere MQ and a transport error (AMRC_TRANSPORT_ERR) will be reported to the application.

Multithreaded applications are not supported on z/OS.

Using WebSphere MQ with the AMI

You must not mix WebSphere MQ function calls with AMI function calls within the same process.

Field limits

When string and binary properties such as queue name, message format, and correlation ID are set, the maximum length values are determined by WebSphere MQ, the underlying message transport. See the rules for naming WebSphere MQ objects in the *WebSphere MQ Application Programming Guide*.

Using the AMI OAMAS subset

A subset of the AMI conforms to the Open Applications Group Middleware Application Programming Interface Specification (OAMAS). See `www.openapplications.org` for further details.

To ensure that your C applications conform to the OAMAS subset, your C functions should include the `oamasami.h` header in place of `amtc.h`.

Building C applications

This section contains information that will help you write, prepare, and run your C application programs on the various operating systems supported by the AMI.

AMI include file

The AMI provides an include file, **amtc.h**, to assist you with the writing of your applications. It is recommended that you become familiar with the contents of this file.

The include file is installed under:

```
QMQMAMI/H       (iSeries)

/amt/inc        (UNIX)

\amt\include    (Windows)

hlq.SCSQC370    (z/OS)
```

See “Directory structure” on page 445 (AIX), page 449 (HP-UX), page 453 (iSeries), page 458 (Solaris), page 462 (Windows), orpage 465 (z/OS).

Your AMI C program must contain the statement:

```
#include <amtc.h>
```

The AMI include file must be accessible to your program at compilation time.

Data types

All data types are defined by means of the **typedef** statement. For each data type, the corresponding pointer data type is also defined. The name of the pointer data type is the name of the elementary or structure data type prefixed with the letter “P” to denote a pointer; for example:

```
typedef AMHSES  AMPOINTER PAMHSES;  /* pointer to AMHSES  */
```

Initial values for structures

The include file amtc.h defines a macro variable that provides initial values for the AMELEM structure. This is the structure used to pass name/value element information across the AMI. Use it as follows:

```
AMELEM MyElement = {AMELEM_DEFAULT};
```

You are recommended to initialize all AMELEM structures in this way so that the `structId` and `version` fields have valid values. If the values passed for these fields are not valid, AMI will reject the structure.

Note that some of the fields in this structure are string pointers that, in the default case, are set to NULL. If you wish to use these fields, you must allocate the correct amount of storage before you set the pointer.

Next step

Now go to one of the following to continue building a C application:
- "C applications on AIX"
- "C applications on HP-UX" on page 33
- "C applications on iSeries" on page 34
- "C applications on Solaris" on page 35
- "C applications on Windows" on page 36
- "C applications on z/OS" on page 36

C applications on AIX

This section explains what you have to do to prepare and run your C programs on the AIX operating system. See *WebSphere MQ for AIX, V5.3 Quick Beginnings* for compilers supported by the AMI.

Preparing C programs on AIX

The following information is not prescriptive, because there are many ways to set up environments to build executables. Use it as a guideline, but follow your local procedures.

To compile an AMI program in a single step using the **xlc** command, you need to specify a number of options:

- Where the AMI include files are.

  To do this, use the `-I` flag. In the case of AIX, they are usually located at `/usr/mqm/amt/inc`.
- Where the AMI library is.

  To do this, use the `-L` flag. In the case of AIX, it is usually located at `/usr/mqm/lib`.
- Link with the AMI library.

  To do this, use the `-l` flag, more specifically `-lamt`.

For example, to compile the C program `mine.c` into an executable called `mine`:

```
xlc -I/usr/mqm/amt/inc -L/usr/mqm/lib -lamt mine.c -o mine
```

If, however, you are building a threaded program, you must use the correct compiler and the threaded library, `libamt_r.a`. For example:

```
xlc_r -I/usr/mqm/amt/inc -L/usr/mqm/lib -lamt_r mine.c -o mine
```

Running C programs on AIX

To run a C executable, you must have access to the C libraries `libamt.a`, `libamtXML310.a`, and `libamtICUUC140.a` in your runtime environment. If the **amtInstall** utility has been run, this environment will be set up for you (see "Installation on AIX" on page 444).

If you have not run the utility, the easiest way of achieving this is to construct a link from the AIX default library location to the actual location of the C libraries. To do this:

```
  ln -s /usr/mqm/lib/libamt.a /usr/lib/libamt.a
  ln -s /usr/mqm/lib/libamtXML310.a /usr/lib/libamtXML310.a
  ln -s /usr/mqm/lib/libamtICUUC140.a /usr/lib/libamtICUUC140.a
```

You must have sufficient access to perform this operation.

If you are using the threaded libraries, you can perform a similar operation:

```
ln -s /usr/mqm/lib/libamt_r.a /usr/lib/libamt_r.a
ln -s /usr/mqm/lib/libamtXML310_r.a /usr/lib/libamtXML310_r.a
ln -s /usr/mqm/lib/libamtICUUC140_r.a /usr/lib/libamtICUUC140_r.a
```

You must also make the AMI WebSphere MQ runtime binding stubs available in your runtime environment. These stubs allow AMI to load WebSphere MQ libraries dynamically.

For the non-threaded WebSphere MQ Server library, perform:

```
ln -s /usr/mqm/lib/amtcmqm /usr/lib/amtcmqm
```

For the non-threaded WebSphere MQ Client library, perform:

```
ln -s /usr/mqm/lib/amtcmqic /usr/lib/amtcmqic
```

For the threaded WebSphere MQ Server library, perform:

```
ln -s /usr/mqm/lib/amtcmqm_r /usr/lib/amtcmqm_r
```

For the threaded WebSphere MQ Client library, perform:

```
ln -s /usr/mqm/lib/amtcmqic_r /usr/lib/amtcmqic_r
```

C applications on HP-UX

This section explains what you have to do to prepare and run your C programs on the HP-UX operating system. See *WebSphere MQ for HP-UX, V5.3 Quick Beginnings* for compilers supported by the AMI.

Preparing C programs on HP-UX

The following information is not prescriptive, because there are many ways to set up environments to build executables. Use it as a guideline, but follow your local procedures.

To compile an AMI program in a single step using the **aCC** command, you need to specify a number of options:

- Where the AMI include files are.

  To do this, use the `-I` flag. In the case of HP-UX, they are usually located at `/opt/mqm/amt/inc`.
- Where the AMI libraries are.

  To do this, use the `-Wl,+b,:,-L` flags. In the case of HP-UX, they are usually located at `/opt/mqm/lib`.
- Link with the AMI library.

  To do this, use the `-l` flag, more specifically `-lamt`.

For example, to compile the AMI C program `mine.c` into an executable called `mine`:

```
aCC +DAportable -Wl,+b,:,-L/opt/mqm/lib -o mine mine.c
      -I/opt/mqm/amt/inc -lamt
```

Note that you could equally link to the threaded library using `-lamt_r`. On HP-UX, there is no difference, because the unthreaded versions of the AMI binaries are simply links to the threaded versions.

C applications on HP-UX

Running C programs on HP-UX

To run a C executable, you must have access to the C libraries `libamt.sl`, `libamtXML310.sl`, and `libamtICUUC140.sl` in your runtime environment. If the **amtInstall** utility has been run, this environment will be set up for you (see "Installation on HP-UX" on page 448).

If you have not run the utility, the easiest way of achieving this is to construct a link from the HP-UX default library location to the actual location of the C libraries. To do this:

```
ln -s /opt/mqm/lib/libamt_r.sl /usr/lib/libamt.sl
ln -s /opt/mqm/lib/libamtXML310_r.sl /usr/lib/libamtXML310.sl
ln -s /opt/mqm/lib/libamtICUUC140_r.sl /usr/lib/libamtICUUC140.sl
```

You must have sufficient access to perform this operation.

If you are using the threaded libraries, you can perform a similar operation:

```
ln -s /opt/mqm/lib/libamt_r.sl /usr/lib/libamt_r.sl
ln -s /opt/mqm/lib/libamtXML310_r.sl /usr/lib/libamtXML310_r.sl
ln -s /opt/mqm/lib/libamtICUUC140_r.sl /usr/lib/libamtICUUC140_r.sl
```

You must also make the AMI WebSphere MQ runtime binding stubs available in your runtime environment. These stubs allow AMI to load WebSphere MQ libraries dynamically.

For the non-threaded WebSphere MQ Server library, perform:

```
ln -s /opt/mqm/lib/amtcmqm_r /usr/lib/amtcmqm
```

For the non-threaded WebSphere MQ Client library, perform:

```
ln -s /opt/mqm/lib/amtcmqic_r /usr/lib/amtcmqic
```

For the threaded WebSphere MQ Server library, perform:

```
ln -s /opt/mqm/lib/amtcmqm_r /usr/lib/amtcmqm_r
```

For the threaded WebSphere MQ Client library, perform:

```
ln -s /opt/mqm/lib/amtcmqic_r /usr/lib/amtcmqic_r
```

As before, note that the unthreaded versions are simply links to the threaded versions.

C applications on iSeries

This section explains what you have to do to prepare and run your C programs on the iSeries system. See *WebSphere MQ for iSeries V5.3 Quick Beginnings* for compilers supported by the AMI.

Preparing C programs on iSeries

The following information is not prescriptive, because there are many ways to set up environments to build executables. Use it as a guideline, but follow your local procedures.

To compile a C module, you can use the OS/400® command **CRTCMOD**. The library QMQMAMI must be in the library list because it contains the amtc.h header file.

You must then bind the output of the compiler with the service program using the **CRTPGM** command. Specify the appropriate AMI service program in the **BDNSRVPGM** option of **CRTPGM**. For example:

```
CRTPGM PGM(pgmname) MODULE(pgmname) BNDSRVPGM(QMQMAMI/AMT)
```

Running C programs on iSeries

When you create your program as described in the previous section, it is bound to the service programs it requires to run. There are no additional runtime requirements.

Alternatively, you might create your program with QMQMAMI in the library list and specify *LIBL for the BNDSRVPGM parameter of CRTPGM. At run time, QMQMAMI must be in the library list.

C applications on Solaris

This section explains what you have to do to prepare and run your C programs in the Sun Solaris operating environment. See *WebSphere MQ for Solaris, V5.3 Quick Beginnings* for compilers supported by the AMI.

Preparing C programs on Solaris

The following information is not prescriptive, because there are many ways to set up environments to build executables. Use it as a guideline, but follow your local procedures.

To compile an AMI program in a single step using the **CC** command, you need to specify a number of options:

- Where the AMI include files are.

  To do this, use the `-I` flag. In the case of Solaris, they are usually located at `/opt/mqm/amt/inc`.
- Where the AMI library is.

  To do this, use the `-L` flag. In the case of Solaris, it is usually located at `/opt/mqm/lib`.
- Link with the AMI library.

  To do this, use the `-l` flag, more specifically `-lamt`.

For example, to compile the C program `mine.c` into an executable called `mine`:

```
  CC -mt -I/opt/mqm/amt/inc -L/opt/mqm/lib -lamt mine.c -o mine
```

Running C programs on Solaris

To run a C executable, you must have access to the C libraries `libamt.so`, `libamtXML310.so`, and `libamtICUUC140.so` in your runtime environment. If the **amtInstall** utility has been run, this environment will be set up for you (see “Installation on Solaris” on page 457).

If you have not run the utility, the easiest way of achieving this is to construct a link from the Solaris default library location to the actual location of the C libraries. To do this:

```
  ln -s /opt/mqm/lib/libamt.so /usr/lib/libamt.so
  ln -s /opt/mqm/lib/libamtXML310.so /usr/lib/libamtXML310.so
  ln -s /opt/mqm/lib/libamtICUUC140.so /usr/lib/libamtICUUC140.so
```

You must have sufficient access to perform this operation.

C applications on Solaris

You must also make the AMI WebSphere MQ runtime binding stubs available in your runtime environment. These stubs allow AMI to load WebSphere MQ libraries dynamically. For the non-threaded WebSphere MQ Server library, perform:

```
ln -s /opt/mqm/lib/amtcmqm /usr/lib/amtcmqm
```

For the WebSphere MQ Client library, perform:

```
ln -s /opt/mqm/lib/amtcmqic /usr/lib/amtcmqic
```

C applications on Windows

This section explains what you have to do to prepare and run your C programs on the Windows 98, Windows NT, Windows Me, and Windows 2000 operating systems. See *WebSphere MQ for Windows, V5.3 Quick Beginnings* for compilers supported by the AMI.

Preparing C programs on Windows

The following information is not prescriptive, because there are many ways to set up environments to build executables. Use it as a guideline, but follow your local procedures.

To compile an AMI program in a single step using the **cl** command, you need to specify a number of options:

- Where the AMI include files are.

  To do this, use the `-I` flag. In the case of Windows, they are usually located at `\amt\include` relative to where you installed WebSphere MQ. Alternatively, the include files could exist in one of the directories pointed to by the INCLUDE environment variable.
- Where the AMI library is.

  To do this, include the library file `amt.LIB` as a command line argument. The `amt.LIB` file should exist in one of the directories pointed to by the LIB environment variable.

For example, to compile the C program `mine.c` into an executable called `mine.exe`:

```
cl -IC:\MQSeries\amt\include /Fomine mine.c amt.LIB
```

Running C programs on Windows

To run a C executable, you must have access to the C DLLs `amt.dll` and `amtXML.dll` in your runtime environment. Make sure they exist in one of the directories pointed to by the PATH environment variable. For example:

```
SET PATH=%PATH%;C:\MQSeries\bin;
```

If you already have WebSphere MQ installed, and you have installed AMI under the WebSphere MQ directory structure, it is likely that the PATH has already been set up for you.

You must also make sure that your AMI runtime environment can access the WebSphere MQ runtime environment. (This will be the case if you installed WebSphere MQ using the documented method.)

C applications on z/OS

This section explains what you have to do to prepare and run your C programs on the z/OS operating system. See *WebSphere MQ for z/OS Concepts and Planning Guide* for compilers supported by the AMI.

Preparing C programs on z/OS

C application programs using the AMI must be compiled, pre-linked, and link edited. Programs containing CICS commands must be processed by the CICS translator before compilation.

Compile: Make sure that the AMI include file (installed in library `hlq.SCSQC370`) is added to the C compiler's SYSLIB concatenation. The C compile options must include DLL and LONGNAME.

Pre-link: The pre-link job step is essential for importing the AMI DLL function references from an appropriate sidedeck. A DD statement for the sidedeck member, `hlq.SCSQDEFS(member)`, must be specified in the pre-link step SYSIN concatenation after the application object code member. The appropriate sidedeck member for each application type is as follows:

Batch	AMTBD10
RRS-batch	AMTRD10
CICS	AMTCD10
IMS	AMTID10

Link Edit: There are no special requirements for link editing.

Running C programs on z/OS

The AMI needs access to the WebSphere MQ datasets SCSQLOAD and SCSQAUTH, as well as one of the language-specific datasets such as SCSQANLE. See the *WebSphere MQ Application Programming Guide* for details of the supported languages. The following list shows which JCL concatenation to add the datasets to for each AMI-supported environment:

Batch	STEPLIB or JOBLIB
CICS	DFHRPL
IMS	The Message Processing Regions' STEPLIB

Writing policy handlers in C

Policy handler libraries are collections of functions that can extend the function of the AMI by performing operations that the AMI does not provide as standard. Examples of such operations include data encryption and decryption, and addition of application-specific message headers.

Policy handler operation is as follows:

1. The AMI creates a policy handler context object at policy creation time by a call to **amPhlCreate**. This function allocates any required memory and returns the context handle.
2. The AMI calls **amPhlInitialize**. This calls back into the AMI, using **amLibRegisterFunction** to register each of the invocation points that the policy handler wishes to support in its implementation.
3. An application calls an AMI function that detects a policy handler invocation point registered for the current policy object (specified on the AMI function call) and AMI operation.
4. The AMI traverses the chain of (one or more) policy handler invocations that were specified in the repository for this policy. It makes a call to run the appropriate policy handler function (**amPhl***Xxx*) for each policy handler with a function for this invocation point.

Error and reason codes can be set in the policy handlers and are accessible to applications in the normal way. See “Error handling” on page 26.

For further details about policy handlers, invocation points, and the functions used for policy handler operation, see Chapter 6, “C policy handler interface” on page 151.

For details about the extended C AMI functions that are provided for use with policy handler libraries, see Appendix C, “Extended C AMI functions” on page 591.

For details about the sample policy handler library that is supplied with the AMI, see “The AMI policy handler sample program (amtsphlr)” on page 487.

The following sections describe how to compile, link and install a policy handler.

Compiling, linking and installing a policy handler

You can compile, link and install a policy handler on AIX, HP-UX, iSeries, Solaris, Windows, or z/OS. For security reasons, policy handler libraries must be located in a predefined directory below the AMI installation directory.

The following C header files are provided for implementing a policy handler library:

amtxc.h	AMI extensions for policy handler callback functions.
amtphlc.h	Policy handler interface definition.
amtphmqc.h	Transport-specific policy handler definitions for WebSphere MQ.

The policy handler library must export the following functions (for example by including these in a definition or export file):

```
amPhlCreate
amPhlInitialize
amPhlDelete
```

The following sections describe how to compile and link a policy handler on specific platforms. The information in these sections is not prescriptive, because there are many ways to set up environments to build dynamic link libraries. Use it for guidance, but follow your local procedures.

Compiling and linking a policy handler on AIX

On AIX, policy handler libraries must be located in the `/usr/mqm/amt/handlers` directory.

The library file name is the name of the policy handler library, as specified using the AMI Administration Tool, plus the prefix `lib` and the suffix `.a` (for a non-threaded library) or `_r.a` (for a threaded library). For example, for the AMI sample policy handler library amtsphlr, the file name is `libamtsphlr.a`.

To compile a policy handler library, first compile the object file with the non-threaded or threaded compiler using the `xlc` or `xlc_r` command with any associated compiler options. Then, use the `ld` command to link the required libraries, also specifying the export file.

To accomplish this, you need to specify a number of options:

- The location of the AMI include file and other include files.

  To do this, use the `-I` flag.

  On AIX, these files are in `/usr/mqm/amt/inc`. Policy handlers such as the AMI policy handler sample program (amtsphlr) also use the MQI include files in `/usr/mqm/tools/c/inc`.
- The location of the AMI library and other libraries.

  To do this, use the `-L` flag.

  On AIX, these libraries are in `/usr/mqm/lib`.
- Link with the AMI library and other libraries.

  To do this, use the `-l` flag, more specifically `-lamt` and, for the WebSphere MQ non-threaded library, `-lmqm` (or `-lmqmic` for a WebSphere MQ client). For a threaded policy handler library, use `-lmqm_r` (or `-lmqmic_r`).

For example, to compile and link the program `amtsphlr.c` as a non-threaded library called `libamtsphlr.a`, use the following commands:

```
xlc -c -I/usr/mqm/inc -I/usr/mqm/amt/inc -o amtsphlr.o amtsphlr.c

ld -e amPhlCreate -o libamtsphlr.a amtsphlr.o -L/usr/mqm/lib -bE:amtsphlr.exp
-lmqm -lamt -lc
```

Alternatively, to compile and link the program `amtsphlr.c` into a threaded library called `libamtsphlr_r.a`, use the following commands:

```
xlc_r -c -I/usr/mqm/inc -I/usr/mqm/amt/inc -o amtsphlr.o amtsphlr.c

ld -e amPhlCreate -o libamtsphlr_r.a amtsphlr.o -L/usr/mqm/lib -bE:amtsphlr.exp
-lpthread -lmqm_r -lamt_r -lc_r
```

Compiling and linking a policy handler on HP-UX

On HP-UX, policy handler libraries must be located in the `/opt/mqm/amt/handlers` directory.

The library file name is the library name of the policy handler, as specified using the AMI Administration Tool, plus the prefix `lib` and the suffix `_r.sl`. For example, for the AMI sample policy handler amtsphlr, the file name is `libamtsphlr_r.sl`.

To compile a policy handler library, first compile the object file using the `aCC` command with any associated compiler options. Then use the `ld` command to link the required libraries, also specifying the export file.

To accomplish this, you need to specify a number of options:

- The location of the AMI include file and other include files.

  To do this, use the `-I` flag.

  On HP-UX, these files are in `/opt/mqm/amt/inc`. Policy handlers such as the AMI policy handler sample program (amtsphlr) also use the MQI include files in `/opt/mqm/tools/c/inc`.
- The location of the AMI library and other libraries.

  To do this, use the `-L` flag.

  On HP-UX, these libraries are in `/opt/mqm/lib`.
- Link with the AMI library and other libraries.

  To do this, use the `-l` flag, more specifically `-lamt_r` and, for the WebSphere MQ library, `-lmqm_r` (or `-lmqmic_r` for a WebSphere MQ client).

For example, to compile and link the program `amtsphlr.c` into a library called `libamtsphlr_r.sl`, use the following commands:

```
aCC +eh -c -I/opt/mqm/inc -I/opt/mqm/amt/inc -O +z -o amtsphlr.o amtsphlr.c

ld -c amtsphlr.exp -b -o libamtsphlr_r.sl +b : -L/opt/mqm/lib amtsphdlr.o -lcl
-lpthread -lc -ldld -lamt_r -lmqm_r -lm -lCsup
```

Compiling and linking a policy handler on iSeries

On iSeries, policy handlers must be service programs (*SRVPGM) located in the QMQMAMIPHL library. The service program name is the library name of the policy handler, as specified using the AMI Administration Tool. The service program name does not require any prefix or suffix.

To create the service program, first you must compile the source with the CRTCMOD command and with the AMI header files, and then you must create the service program with the CRTSRVPGM command binding to the appropriate AMI and WebSphere MQ libraries.

To do this, use the following steps:

1. Make the AMI header files accessible.

   To do this, use ADDLIBLE to add the AMI header files to your library list.
2. Compile the sample.

   Use the CRTCMOD command to create a module.
3. Create the service program.

   Use the CRTSRVPGM command to create a service program. You must bind to the appropriate WebSphere MQ and AMI libraries. If you are using the non-threaded libraries, bind to AMT and LIBMQM. If you are using threaded libraries, bind to AMT_R and LIBMQM_R .

For example, to create a non threaded version of the sample service program AMTSPHLR in library QMQMAMIPHL, use the following commands:

```
ADDLIBLE QMQMAMI
ADDLIBLE QMQM

CRTCMOD MODULE(QMQMAMI/AMTSPHLR) SRCFILE(QMQMAMI/QCSRC) SRCMBR(AMTSPHLR)

CRTSRVPGM SRVPGM(QMQMAMIPHL/AMTSPHLR) MODULE(QMQMAMI/AMTSPHLR)
    EXPORT(*SRCFILE )SRCMBR(AMTSPHLR) BNDSRVPGM(QMQMAMI/AMT QMQM/LIBMQM)
    ACTGRP(*CALLER)
```

Compiling and linking a policy handler on Solaris

On Solaris, policy handler libraries must be located in the `/opt/mqm/amt/handlers` directory.

The library file name is the library name of the policy handler, as specified using the AMI Administration Tool, plus the prefix `lib` and the suffix `.so`. For example, for the AMI sample policy handler amtsphlr, the file name is `libamtsphlr.so`.

To compile a policy handler library, first compile the object file using the `CC` command with any associated compiler options. Then, use the `CC` command to link the required libraries, also specifying the export file.

To accomplish this, you need to specify a number of options:

- The location of the AMI include file and other include files.

  To do this, use the `-I` flag.

  On Sun Solaris, these files are in `/opt/mqm/amt/inc`. Policy handlers such as the AMI policy handler sample program (amtsphlr) also use the MQI include files in `/opt/mqm/tools/c/inc`.
- The location of the AMI library and other libraries.

  To do this, use the `-L` flag.

  On Sun Solaris, these libraries are in `/opt/mqm/lib`.
- Link with the AMI library and other libraries.

  To do this, use the `-l` flag, more specifically `-lamt` and, for the WebSphere MQ library, `-lmqm` (or `-lmqmic` for a WebSphere MQ client).

For example, to compile and link the program `amtsphlr.c` into a threaded library called `libamtsphlr.so`, use the following commands:

```
CC -c -I/opt/mqm/inc -I/opt/mqm/amt/inc -o amtsphlr.o amtsphlr.c

CC -mt -o libamtsphlr.so -L/opt/mqm/lib -dy -ldl -lamt -lmqm -G amtsphlr.o
```

Compiling and linking a policy handler on Windows

On Windows, policy handler libraries must be located in the `amt\handlers` directory.

To compile an AMI policy handler library using the `cl` command, you need to specify a number of options:

- The location of the AMI include file and other include files .

  To do this, use the `/I` flag. On Windows, these files are usually in `.\amt\include`, relative to where you installed WebSphere MQ. Policy handlers such as the AMI sample policy handler program (amtsphlr) also use the MQI include files, usually in `.\tools\c\include`, relative to where you installed WebSphere MQ.

- The location of the AMI library and other libraries.

  To do this, include the library file `amt.lib` plus any other library files (e.g. `mqm.lib` or `mqmic32.lib`) as command line arguments. These library files must be a directory that is specified by the `lib` environment variable.

Also, ensure that directory `.\MQSeries\tools\lib` is included in your LIB environment variable.

For example, if WebSphere MQ is installed in the C:\ directory, to compile the program `amtsphlr.c` into a library called `amtsphlr.dll`, use the following commands:

```
cl -c -Ic:\mqseries\amt\include -Ic:\mqseries\tools\c\include amtsphlr.c

lib -out:amtsphlr.LIB -def:amtsphlr.def -machine:IX86

link -nod -nologo -dll amtsphlr.obj amtsphlr.exp amt.lib mqm.lib msvcrt.lib
oldnames.lib kernel32.lib ws2_32.lib mswsock.lib advapi32.lib user32.lib
netapi32.lib -out:amtsphlr.dll
```

Compiling, prelinking, and linking a policy handler on z/OS

A policy handler is built as a C dynamic load library and is stored as a member in a PDS on z/OS. A policy handler is prepared according to the z/OS environment in which it is to run, in the same way as an AMI application program (see “C applications on z/OS” on page 36).

Compile: The C compile options must include DLL, LO, NOSEQ, and RENT.

Pre-link: The pre-link job step is essential for importing the AMI DLL function references from an appropriate sidedeck. A DD statement for the sidedeck member, `hlq.SCSQDEFS(member)`, must be specified in the pre-link step SYSIN concatenation after the policy handler object code member. Use the appropriate sidedeck member according to the z/OS environment in which the policy handler is to be used, as follows:

Batch	AMTBD10
RRS-batch	AMTRD10
CICS	AMTCD10
IMS	AMTID10

The policy handler code contains its own exported functions, so that as a by-product, the pre-link produces an output sidedeck. Therefore, a DD statement for SYSDEFSD must be supplied, although this may be specified as DUMMY, because this dataset is not required.

Link Edit: The link edit options must include RENT and DYNAM(DLL).

Chapter 3. The C high-level interface

The C high-level interface contains functions that cover the requirements of most applications. If extra functionality is needed, C object interface functions can be used in the same application as the C high-level functions.

This chapter contains:

- “Overview of the C high-level interface” on page 44
- “Reference information for the C high-level interface” on page 45

Overview of the C high-level interface

The following section lists the high-level functions. Follow the page references to see the detailed descriptions of each function.

Initialize and terminate

Functions to create and open an AMI session, and to close and delete an AMI session.

amInitialize	page 51
amTerminate	page 66

Sending messages

Functions to send a datagram (send and forget) message, and to send request and response messages.

amSendMsg	page 62
amSendRequest	page 63
amSendResponse	page 64

Receiving messages

Functions to receive a message from **amSendMsg** or **amSendResponse**, and to receive a request message from **amSendRequest**.

amReceiveMsg	page 55
amReceiveRequest	page 59
amBrowseMsg	page 48

File transfer

Functions to send message data from a file, and to receive message data sent by **amSendFile** into a file.

amSendFile	page 61
amReceiveFile	page 53

Publish/subscribe

Functions to publish a message to a publish/subscribe broker, and to subscribe, unsubscribe, and receive publications.

amPublish	page 52
amSubscribe	page 65
amUnsubscribe	page 67
amReceivePublication	page 57

Transaction support

Functions to begin, commit, and back out a unit of work.

amBegin	page 47
amCommit	page 50
amBackout	page 46

Reference information for the C high-level interface

In the following sections the high-level interface functions are listed in alphabetical order. Note that all functions return a completion code (`pCompCode`) and a reason code (`pReason`). The completion code can take one of the following values:

AMCC_OK	Function completed successfully
AMCC_WARNING	Function completed with a warning
AMCC_FAILED	An error occurred during processing

If the completion code returns warning or failed, the reason code identifies the reason for the error or warning (see Appendix A, "Reason codes and LDAP error codes" on page 537).

Most functions require the session handle to be specified. If this handle is not valid, the results are unpredictable.

amBackout

Function to back out a unit of work.

```
AMBOOL amBackout(
  AMHSES    hSession,
  AMSTR     policyName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amBegin

Function to begin a unit of work.

```
AMBOOL amBegin(
  AMHSES    hSession,
  AMSTR     policyName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amBrowseMsg

Function to browse a message. See the *WebSphere MQ Application Programming Guide* for a full description of the browse options.

```
AMBOOL amBrowseMsg(
  AMHSES    hSession,
  AMSTR     receiverName,
  AMSTR     policyName,
  AMLONG    options,
  AMLONG    buffLen,
  PAMLONG   pDataLen,
  PAMBYTE   pData,
  AMSTR     rcvMsgName,
  AMSTR     senderName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

`hSession`	The session handle returned by **amInitialize** (input).
`receiverName`	The name of a receiver service (input). If specified as NULL, the system default receiver name (constant: AMSD_RCV) is used.
`policyName`	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
`options`	Options controlling the browse operation (input). Possible values are:

```
AMBRW_NEXT
AMBRW_FIRST
AMBRW_CURRENT
AMBRW_RECEIVE_CURRENT
AMBRW_DEFAULT          (AMBRW_NEXT)
AMBRW_LOCK_NEXT        (AMBRW_LOCK + AMBRW_NEXT)
AMBRW_LOCK_FIRST       (AMBRW_LOCK + AMBRW_FIRST)
AMBRW_LOCK_CURRENT     (AMBRW_LOCK + AMBRW_CURRENT)
AMBRW_UNLOCK
```

`AMBRW_RECEIVE_CURRENT` is equivalent to **amRcvReceive** for the message under the browse cursor.

Note that a locked message is unlocked by another browse or receive, even though it is not for the same message.

`buffLen`	The length in bytes of a buffer in which the data is returned (input).
`pDataLen`	The length of the message data, in bytes (output). Specify as NULL if this is not required.
`pData`	The received message data (output).
`rcvMsgName`	The name of the message object for the received message (output). Properties, and message data if not returned in the `pData` parameter, can be extracted from the message object using the object interface (see “Message interface functions” on page 96). The message object is implicitly reset before the browse takes place. If `rcvMsgName` is specified as NULL, the system default receive message name (constant: AMSD_RCV_MSG) is used.

`senderName`	The name of a special type of sender service known as a *response sender*, to which the response message will be sent (output). This sender name must not be defined in the repository. It is only applicable if the message type is AMMT_REQUEST. Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.
`pCompCode`	Completion code (output).
`pReason`	Reason code (output).

Usage notes

You can return the message data in the message object or in an application buffer.

To return the data in the message object (`rcvMsgName`), set `buffLen` to zero, and set `pData` and `pDataLen` to values that are not NULL.

To return data in an application message buffer:
- set `pData` to the buffer pointer value (that is, not NULL)
- set `buffLen` to the length of the buffer

If the value of `buffLen` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a value for `pDataLen` that is not NULL.

To return only the data length:
- set `pData` to NULL
- set `buffLen` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

amCommit

Function to commit a unit of work.

```
AMBOOL amCommit(
  AMHSES    hSession,
  AMSTR     policyName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amInitialize

Function to create and open an AMI session. It returns a session handle of type AMHSES, which is valid until the session is terminated. One **amInitialize** is allowed per thread. A session handle can be used on different threads, subject to any limitations of the underlying transport layer (WebSphere MQ).

```
AMHSES amInitialize(
  AMSTR     name,
  AMSTR     policyName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

name	An optional name that can be used to identify the application (input).
policyName	The name of a policy defined in the repository (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amPublish

Function to publish a message to a publish/subscribe broker.

```
AMBOOL amPublish(
  AMHSES    hSession,
  AMSTR     publisherName,
  AMSTR     policyName,
  AMSTR     responseName,
  AMLONG    topicLen,
  AMSTR     pTopic,
  AMLONG    dataLen,
  PAMBYTE   pData,
  AMSTR     pubMsgName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
publisherName	The name of a publisher service (input). If specified as NULL, the system default publisher name (constant: AMSD_PUB) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
responseName	The name of the receiver service to which the response to this publish request should be sent (input). Specify as NULL if no response is required. This parameter is mandatory if the policy specifies implicit publisher registration (the default).
topicLen	The length of the topic for this publication, in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is NULL terminated.
pTopic	The topic for this publication (input).
dataLen	The length of the publication data in bytes (input). A value of zero indicates that any publication data has been added to the message object (`pubMsgName`) using the object interface (see “Message interface functions” on page 96).
pData	The publication data, if `dataLen` is non-zero (input).
pubMsgName	The name of a message object that contains the header for the publication message (input). If `dataLen` is zero,. it also holds any publication data. If specified as NULL, the system default message name (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amReceiveFile

Function to receive message data sent by **amSendFile** into a file.

```
AMBOOL amReceiveFile(
  AMHSES    hSession,
  AMSTR     receiverName,
  AMSTR     policyName,
  AMLONG    options,
  AMSTR     selMsgName,
  AMLONG    directoryLen,
  AMSTR     directory,
  AMLONG    fileNameLen,
  AMSTR     fileName,
  AMSTR     rcvMsgName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
receiverName	The name of a receiver service (input). If specified as NULL, the system default receiver name (constant: AMSD_RCV) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
options	A reserved field that must be specified as zero (input).
selMsgName	Optional selection message object used to specify information (such as a *CorrelId*) needed to select the required message (input).
directoryLen	A reserved field that must be specified as zero (input).
directory	A reserved field that must be specified as NULL (input).
fileNameLen	The length of the file name in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
fileName	The name of the file into which the transferred data is to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If NULL or a null string is specified, the AMI will use the name of the originating file (including any directory prefix), exactly as it was supplied on the send file call. Note that the original file name may not be appropriate for use by the receiver, either because a path name included in the file name is not applicable to the receiving system, or because the sending and receiving systems use different file name conventions.
rcvMsgName	The name of the message object to be used to receive the file (output). This parameter is updated with the message properties (for example, the Message ID). If the message is not from a file, `rcvMsgName` receives the message data. If specified as NULL, the system default receive message name (constant AMSD_RCV_MSG) is used. Property information and message data can be extracted from the message object using the object interface (see "Message interface functions" on page 96). The message object is reset implicitly before the receive takes place.
pCompCode	Completion code (output).

`pReason`	Reason code (output).

Usage notes

If `fileName` is blank (indicating that the originating file name specified in the message is to be used), `fileNameLen` should be set to zero.

amReceiveMsg

Function to receive a message.

```
AMBOOL amReceiveMsg(
  AMHSES    hSession,
  AMSTR     receiverName,
  AMSTR     policyName,
  AMSTR     selMsgName,
  AMLONG    buffLen,
  PAMLONG   pDataLen,
  PAMBYTE   pData,
  AMSTR     rcvMsgName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

`hSession`	The session handle returned by **amInitialize** (input).
`receiverName`	The name of a receiver service (input). If specified as NULL, the system default receiver name (constant: AMSD_RCV) is used.
`policyName`	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
`selMsgName`	Optional selection message object used to specify information (such as a *`CorrelId`*) needed to select the required message (input).
`buffLen`	The length in bytes of a buffer in which the data is returned (input).
`pDataLen`	The length of the message data, in bytes (output). Specify as NULL if this is not required.
`pData`	The received message data (output).
`rcvMsgName`	The name of the message object for the received message (output). If specified as NULL, the system default receive message name (constant: AMSD_RCV_MSG) is used. Properties, and message data if not returned in the `pData` parameter, can be extracted from the message object using the object interface (see “Message interface functions” on page 96). The message object is implicitly reset before the receive takes place.
`pCompCode`	Completion code (output).
`pReason`	Reason code (output).

Usage notes

You can return the message data in the message object or in an application buffer.

To return the data in the message object (`rcvMsgName`), set `buffLen` to zero, and set `pData` and `pDataLen` to values that are not NULL.

To return data in an application message buffer:
- set `pData` to the buffer pointer value (that is, not NULL)
- set `buffLen` to the length of the buffer

If the value of `buffLen` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a value for `pDataLen` that is not NULL.

To return only the data length without removing the message from the queue:

- set `pData` to NULL
- set `buffLen` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

To remove the message from the queue and discard it:

- set `pData` or `pDataLen` to a value that is not NULL
- set `buffLen` to zero
- ensure that Accept Truncated Message in the policy receive attributes is selected

The message will be discarded with an AMRC_MSG_TRUNCATED warning.

If AMRC_RECEIVE_BUFF_LEN_ERR is returned, the message length value is returned in `pDataLen` (if it is not NULL), even though the completion code is MQCC_FAILED.

Note that if `pData` is NULL and `buffLen` is not zero, there is always an AMRC_RECEIVE_BUFF_LEN_ERR error.

amReceivePublication

Function to receive a publication from a publish/subscribe broker.

```
AMBOOL amReceivePublication(
  AMHSES    hSession,
  AMSTR     subscriberName,
  AMSTR     policyName,
  AMSTR     selMsgName,
  AMLONG    topicBuffLen,
  AMLONG    buffLen,
  PAMLONG   pTopicCount,
  PAMLONG   pTopicLen,
  AMSTR     pFirstTopic,
  PAMLONG   pDataLen,
  PAMBYTE   pData,
  AMSTR     rcvMsgName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
subscriberName	The name of a subscriber service (input). If specified as NULL, the system default subscriber name (constant: AMSD_SUB) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
selMsgName	Optional selection message object used to specify information (such as a *CorrelId*) needed to select the required message (input).
topicBuffLen	The length in bytes of a buffer in which the topic is returned (input).
buffLen	The length in bytes of a buffer in which the publication data is returned (input).
pTopicCount	The number of topics in the message (output). Specify as NULL if this is not required.
pTopicLen	The length in bytes of the first topic (output). Specify as NULL if this is not required.
pFirstTopic	The first topic (output). Specify as NULL if this is not required. Topics can be extracted from the message object (`rcvMsgName`) using the object interface (see “Message interface functions” on page 96).
pDataLen	The length in bytes of the publication data (output). Specify as NULL if this is not required.
pData	The publication data (output). Specify as NULL if this is not required. Data can be extracted from the message object (`rcvMsgName`) using the object interface (see “Message interface functions” on page 96).
rcvMsgName	The name of a message object for the received message (input). If specified as NULL, the default message name (constant: AMSD_RCV_MSG) is used. The publication message properties and data update this message object, in addition to being returned in the parameters above. The message object is implicitly reset to the default before the receive takes place.

C high-level interface

`pCompCode`	Completion code (output).
`pReason`	Reason code (output).

Usage notes

We recommend that, when using **amReceivePublication**, you always have data conversion enabled in the specified policy. If data conversion is not enabled, **amReceivePublication** will fail if the local CCSID and/or encoding values differ from those on the platform from which the publication was sent.

If data conversion is enabled by the specified policy, and a selection message is specified, the conversion is performed using the target encoding and coded character set identifier (CCSID) values designated in the selection message. (The selection message is specified in the selMsgName parameter).

If a selection message is not specified, the platform encoding and Queue Manager CCSID values are used as defaults for the conversion.

If a normal message that is not a publication message is received by the specified subscriber, **amReceivePublication** behaves the same as **amReceiveMsg**.

amReceiveRequest

Function to receive a request message.

```
AMBOOL amReceiveRequest(
  AMHSES    hSession,
  AMSTR     receiverName,
  AMSTR     policyName,
  AMLONG    buffLen,
  PAMLONG   pDataLen,
  PAMBYTE   pData,
  AMSTR     rcvMsgName,
  AMSTR     senderName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
receiverName	The name of a receiver service (input). If specified as NULL, the system default receiver name (constant: AMSD_RCV) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
buffLen	The length in bytes of a buffer in which the data is returned (input).
pDataLen	The length of the message data, in bytes (output). Specify as NULL if this is not required.
pData	The received message data (output).
rcvMsgName	The name of the message object for the received message (output). If specified as NULL, the system default receiver service (constant: AMSD_RCV_MSG) is used. Header information, and message data if not returned in the `Data` parameter, can be extracted from the message object using the object interface (see “Message interface functions” on page 96). The message object is implicitly reset before the receive takes place.
senderName	The name of a special type of sender service known as a *response sender*, to which the response message will be sent (output). This sender name must not be defined in the repository. If specified as NULL, the system default response sender service (constant: AMSD_RSP_SND) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

The following notes contain details about use of the **amReceiveRequest** call.

Data conversion

If data conversion is enabled by the specified policy, and a selection message is specified, the conversion is performed using the target encoding and coded character set identifier (CCSID) values designated in the selection message. (These target values are specified in the selMsgName parameter).

If a selection message is not specified, the platform encoding and Queue Manager CCSID values are used as defaults for conversion.

Use of the buffLen parameter

You can return the message data in the message object or in an application buffer.

To return the data in the message object (`rcvMsgName`), set `buffLen` to zero, and set `pData` and `pDataLen` to values that are not NULL.

To return data in an application message buffer:

- set `pData` to the buffer pointer value (that is, not NULL)
- set `buffLen` to the length of the buffer

If the value of `buffLen` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a value for `pDataLen` that is not NULL.

To return only the data length without removing the message from the queue:

- set `pData` to NULL
- set `buffLen` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

To remove the message from the queue and discard it:

- set `pData` or `pDataLen` to a value that is not NULL
- set `buffLen` to zero
- ensure that Accept Truncated Message in the policy receive attributes is selected

The message will be discarded with an AMRC_MSG_TRUNCATED warning.

If AMRC_RECEIVE_BUFF_LEN_ERR is returned, the message length value is returned in `pDataLen` (if it is not NULL), even though the completion code is MQCC_FAILED.

Note that if `pData` is NULL and `buffLen` is not zero, there is always an AMRC_RECEIVE_BUFF_LEN_ERR error.

amSendFile

Function to send data from a file.

```
AMBOOL amSendFile(
  AMHSES    hSession,
  AMSTR     senderName,
  AMSTR     policyName,
  AMLONG    options,
  AMLONG    directoryLen,
  AMSTR     directory,
  AMLONG    fileNameLen,
  AMSTR     fileName,
  AMSTR     sndMsgName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
senderName	The name of a sender service (input). If specified as NULL, the system default sender name (constant: AMSD_SND) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
options	A reserved field that must be specified as zero (input).
directoryLen	A reserved field that must be specified as zero (input).
directory	A reserved field that must be specified as NULL (input).
fileNameLen	The length of the file name in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
fileName	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with a receive file call (see “amReceiveFile” on page 53 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
sndMsgName	The name of the message object to be used to send the file (input). This parameter can be used, for example, to specify the Correlation ID, which can be set from the message object using the object interface (see “Message interface functions” on page 96).
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

The message object is implicitly reset by the **amSendFile call**.

The system default object is used when you set `sndMsgName` to NULL or an empty string.

amSendMsg

Function to send a datagram (send and forget) message.

```
AMBOOL amSendMsg(
  AMHSES    hSession,
  AMSTR     senderName,
  AMSTR     policyName,
  AMLONG    dataLen,
  PAMBYTE   pData,
  AMSTR     sndMsgName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
senderName	The name of a sender service (input). If specified as NULL, the system default sender name (constant: AMSD_SND) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
dataLen	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (`sndMsgName`) using the object interface (see “Message interface functions” on page 96).
pData	The message data, if `dataLen` is non-zero (input).
sndMsgName	The name of a message object for the message being sent (input). If `dataLen` is zero it also holds any message data. If specified as NULL, the system default message name (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSendRequest

Function to send a request message.

```
AMBOOL amSendRequest(
  AMHSES    hSession,
  AMSTR     senderName,
  AMSTR     policyName,
  AMSTR     responseName,
  AMLONG    dataLen,
  PAMBYTE   pData,
  AMSTR     sndMsgName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
senderName	The name of a sender service (input). If specified as NULL, the system default sender name (constant: AMSD_SND) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy (constant: AMSD_POL) is used.
responseName	The name of the receiver service to which the response to this send request should be sent (input). See **amReceiveRequest**. Specify as NULL if no response is required.
dataLen	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (`sndMsgName`) using the object interface (see “Message interface functions” on page 96).
pData	The message data, if `dataLen` is non-zero (input).
sndMsgName	The name of a message object for the message being sent (input). If specified as NULL, the system default message (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSendResponse

Function to send a response to a request message.

```
AMBOOL amSendResponse(
  AMHSES    hSession,
  AMSTR     senderName,
  AMSTR     policyName,
  AMSTR     rcvMsgName,
  AMLONG    dataLen,
  PAMBYTE   pData,
  AMSTR     sndMsgName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
senderName	The name of the sender service (input). It must be set to the `senderName` specified for the **amReceiveRequest** function.
policyName	The name of a policy (input). If specified as NULL, the system default policy (constant: AMSD_POL) is used.
rcvMsgName	The name of the received message that this message is a response to (input). It must be set to the `rcvMsgName` specified for the **amReceiveRequest** function.
dataLen	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (`sndMsgName`) using the object interface (see “Message interface functions” on page 96).
pData	The message data, if `dataLen` is non-zero (input).
sndMsgName	The name of a message object for the message being sent (input). If specified as NULL, the system default message (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubscribe

Function to register a subscription with a publish/subscribe broker. Publications matching the subscription are sent to the receiver service associated with the subscriber. By default, this has the same name as the subscriber service, with the addition of the suffix '.RECEIVER'.

Subscribing applications can exploit content based publish/subscribe by passing a filter on the **amSubscribe** call.

```
AMBOOL amSubscribe(
  AMHSES    hSession,
  AMSTR     subscriberName,
  AMSTR     policyName,
  AMSTR     responseName,
  AMLONG    topicLen,
  AMSTR     pTopic,
  AMLONG    filterLen,
  AMSTR     pFilter,
  AMSTR     subMsgName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
subscriberName	The name of a subscriber service (input). If specified as NULL, the system default subscriber (constant: AMSD_SUB) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy (constant: AMSD_POL) is used.
responseName	The name of the receiver service to which the response to this subscribe request should be sent (input). Specify as NULL if no response is required. This is not the service to which publications will be sent by the broker; they are sent to the receiver service associated with the subscriber (see above).
topicLen	The length of the topic for this subscription, in bytes (input).
pTopic	The topic for this subscription (input). Publications which match this topic, including wildcards, will be sent to the subscriber. Multiple topics can be specified in the message object (`subMsgName`) using the object interface (see "Message interface functions" on page 96).
filterLen	The length in bytes of the filter (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
pFilter	The filter to be added (input). The syntax of the filter string is described in the *WebSphere MQ Integrator Version 2.0 Programming Guide*.
subMsgName	The name of a message object for the subscribe message (input). If specified as NULL, the system default message (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amTerminate

Closes the session, closes and deletes any implicitly created objects, and deletes the session. Any outstanding units of work are committed (if the application terminates without an **amTerminate** call being issued, any outstanding units of work are backed out).

```
AMBOOL amTerminate(
  PAMHSES   phSession,
  AMSTR     policyName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

phSession	A *pointer* to the session handle returned by **amInitialize** (input/output).
policyName	The name of a policy (input). If specified as NULL, the system default policy (constant: AMSD_POL) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amUnsubscribe

Function to remove a subscription from a publish/subscribe broker.

```
AMBOOL amUnsubscribe(
  AMHSES    hSession,
  AMSTR     subscriberName,
  AMSTR     policyName,
  AMSTR     responseName,
  AMLONG    topicLen,
  AMSTR     pTopic,
  AMLONG    filterLen,
  AMSTR     pFilter,
  AMSTR     unsubMsgName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by **amInitialize** (input).
subscriberName	The name of a subscriber service (input). If specified as NULL, the system default subscriber (constant: AMSD_SUB) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy (constant: AMSD_POL) is used.
responseName	The name of the receiver service to which the response to this unsubscribe request should be sent (input). Specify as NULL if no response is required.
topicLen	The length of the topic, in bytes (input).
pTopic	The topic that identifies the subscription to be removed (input). Multiple topics can be specified in the message object (`unsubMsgName`) using the object interface (see "Message interface functions" on page 96). To deregister all topics, a policy that provides this option must be specified (this is not the default policy). Otherwise, to remove a previous subscription, the topic information specified must match that specified on the relevant **amSubscribe** request.
filterLen	The length in bytes of the filter (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
pFilter	The filter that identifies the subscription to be removed (input). The syntax of the filter string is described in the *WebSphere MQ Integrator Version 2.0 Programming Guide*.
unsubMsgName	The name of a message object for the unsubscribe message (input). If specified as NULL, the system default message (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

To successfully remove a previous subscription, you must ensure that the topic, filter, and subscriber queue information exactly matches that used on the original subscribe request.

Chapter 4. C object interface overview

This chapter contains an overview of the structure of the C object interface. Use it to find out what functions are available in this interface.

The object interface provides sets of interface functions for each of the following objects:

Session	page 70
Message	page 72
Sender	page 74
Receiver	page 75
Distribution list	page 76
Publisher	page 77
Subscriber	page 78
Policy	page 79

These interface functions are invoked as necessary by the high-level functions. They are made available to the application programmer through this object-style interface to provide additional function where needed. An application program can mix high-level functions and object-interface functions as required.

Details of the interface functions for each object are given in the following pages. Follow the page references to see the detailed descriptions of each function.

Details of the object interface functions used by each high-level function are given on page 80.

Session interface functions

The session object creates and manages all other objects, and provides the scope for a unit of work.

Session management

Functions to create, open, close, and delete a session object.

amSesCreate	page 85
amSesOpen	page 94
amSesClose	page 85
amSesDelete	page 89

Create objects

Functions to create message, sender, receiver, distribution list, publisher, subscriber, and policy objects. Handles to these objects are returned by these functions.

amSesCreateMessage	page 86
amSesCreateSender	page 88
amSesCreateReceiver	page 87
amSesCreateDistList	page 86
amSesCreatePublisher	page 87
amSesCreateSubscriber	page 88
amSesCreatePolicy	page 86

Get object handles

Functions to get the handles for a message, sender, receiver, distribution list, publisher, subscriber, and policy objects with a specified name (needed if the objects were created implicitly by the high-level interface).

amSesGetMessageHandle	page 92
amSesGetSenderHandle	page 94
amSesGetReceiverHandle	page 93
amSesGetDistListHandle	page 91
amSesGetPublisherHandle	page 93
amSesGetSubscriberHandle	page 94
amSesGetPolicyHandle	page 93

Delete objects

Functions to delete message, sender, receiver, distribution list, publisher, subscriber, and policy objects.

amSesDeleteMessage	page 89
amSesDeleteSender	page 91
amSesDeleteReceiver	page 90
amSesDeleteDistList	page 89
amSesDeletePublisher	page 90
amSesDeleteSubscriber	page 91
amSesDeletePolicy	page 90

Transactional processing

Functions to begin, commit, and rollback a unit of work.

amSesBegin	page 84
amSesCommit	page 85
amSesRollback	page 95

Error handling

Functions to clear the error codes, and return the completion and reason codes for the last error associated with the session object.

amSesClearErrorCodes	page 84
amSesGetLastError	page 92

Message interface functions

A message object encapsulates a WebSphere MQ message descriptor (MQMD) structure. It also contains the message data if this is not passed as a separate parameter.

Get values

Functions to get the coded character set ID, correlation ID, encoding, format, group status, message ID, and name of the message object.

amMsgGetCCSID	page 100
amMsgGetCorrelId	page 100
amMsgGetElementCCSID	page 102
amMsgGetEncoding	page 102
amMsgGetFormat	page 104
amMsgGetGroupStatus	page 104
amMsgGetMsgId	page 105
amMsgGetName	page 106
amMsgGetReportCode	page 107
amMsgGetType	page 108

Set values

Functions to set the coded character set ID, correlation ID, encoding, format, group status, feedback code type, and message type of the message object.

amMsgSetCCSID	page 110
amMsgSetCorrelId	page 110
amMsgSetElementCCSID	page 111
amMsgSetEncoding	page 111
amMsgSetFormat	page 112
amMsgSetGroupStatus	page 112
amMsgSetReportCode	page 113
amMsgSetType	page 113

Reset values

Function to reset the message object to the state it had when first created.

amMsgReset	page 109

Read and write data

Functions to get the length of the data, get and set the data offset, and read or write byte data to or from the message object at the current offset.

amMsgGetDataLength	page 101
amMsgGetDataOffset	page 101
amMsgSetDataOffset	page 110

amMsgReadBytes	page 109
amMsgWriteBytes	page 114

Publish/subscribe topics

Functions to manipulate the topics in a publish/subscribe message.

amMsgAddTopic	page 98
amMsgDeleteTopic	page 100
amMsgGetTopic	page 108
amMsgGetTopicCount	page 109

Publish/subscribe filters

Functions to manipulate the filters in a publish/subscribe message.

amMsgAddFilter	page 97
amMsgDeleteFilter	page 99
amMsgGetFilter	page 103
amMsgGetFilterCount	page 103

Publish/subscribe name/value elements

Functions to manipulate the name/value elements in a publish/subscribe message.

amMsgAddElement	page 97
amMsgDeleteElement	page 98
amMsgGetElement	page 101
amMsgGetElementCount	page 102
amMsgDeleteNamedElement	page 99
amMsgGetNamedElement	page 106
amMsgGetNamedElementCount	page 107

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the message.

amMsgClearErrorCodes	page 98
amMsgGetLastError	page 105

Publish/subscribe helper macros

Helper macros provided for use with the publish/subscribe stream name and publication timestamp name/value strings.

AmMsgAddStreamName	page 115
AmMsgGetPubTimestamp	page 115
AmMsgGetStreamName	page 116

Sender interface functions

A sender object encapsulates a WebSphere MQ object descriptor (MQOD) structure for sending a message.

Open and close

Functions to open and close the sender service.

amSndOpen	page 119
amSndClose	page 117

Send

Function to send a message.

amSndSend	page 120
amSndSendFile	page 121

Get values

Functions to get the coded character set ID, encoding, and name of the sender service.

amSndGetCCSID	page 118
amSndGetEncoding	page 118
amSndGetName	page 119

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the sender service.

amSndClearErrorCodes	page 117
amSndGetLastError	page 118

Receiver interface functions

A receiver object encapsulates a WebSphere MQ object descriptor (MQOD) structure for receiving a message.

Open and close

Functions to open and close the receiver service.

amRcvOpen	page 128
amRcvClose	page 126

Receive and browse

Functions to receive or browse a message.

amRcvReceive	page 129
amRcvReceiveFile	page 131
amRcvBrowse	page 122
amRcvBrowseSelect	page 124

Get values

Functions to get the definition type, name, and queue name of the receiver service.

amRcvGetDefnType	page 126
amRcvGetName	page 127
amRcvGetQueueName	page 128

Set values

Function to set the queue name of the receiver service.

amRcvSetQueueName	page 132

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the receiver service.

amRcvClearErrorCodes	page 125
amRcvGetLastError	page 127

Distribution list interface functions

A distribution list object encapsulates a list of sender services.

Open and close

Functions to open and close the distribution list service.

amDstOpen	page 135
amDstClose	page 133

Send

Function to send a message to the distribution list.

amDstSend	page 136
amDstSendFile	page 137

Get values

Functions to get the name of the distribution list service, a count of the sender services in the list, and a sender service handle.

amDstGetName	page 134
amDstGetSenderCount	page 135
amDstGetSenderHandle	page 135

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the distribution list.

amDstClearErrorCodes	page 133
amDstGetLastError	page 134

Publisher interface functions

A publisher object encapsulates a sender service. It provides support for publishing messages to a publish/subscribe broker.

Open and close

Functions to open and close the publisher service.

amPubOpen	page 140
amPubClose	page 138

Publish

Function to publish a message.

amPubPublish	page 141

Get values

Functions to get the coded character set ID, encoding, and name of the publisher service.

amPubGetCCSID	page 138
amPubGetEncoding	page 139
amPubGetName	page 140

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the publisher.

amPubClearErrorCodes	page 138
amPubGetLastError	page 139

Subscriber interface functions

A subscriber object encapsulates both a sender service and a receiver service. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

Open and close

Functions to open and close the subscriber service.

amSubOpen	page 145
amSubClose	page 142

Broker messages

Functions to subscribe to a broker, remove a subscription, and receive publications from the broker.

amSubSubscribe	page 147
amSubUnsubscribe	page 148
amSubReceive	page 146

Get values

Functions to get the coded character set ID, definition type, encoding, name, and queue name of the subscriber service.

amSubGetCCSID	page 142
amSubGetDefnType	page 143
amSubGetEncoding	page 143
amSubGetName	page 144
amSubGetQueueName	page 145

Set value

Function to set the queue name of the subscriber service.

amSubSetQueueName	page 146

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the receiver.

amSubClearErrorCodes	page 142
amSubGetLastError	page 144

Policy interface functions

A policy object encapsulates details of how the message is handled (such as its priority, its persistence, and whether it is included in a unit of work).

Get values

Functions to get the name of the policy, and the wait time set in the policy.

amPolGetName	page 150
amPolGetWaitTime	page 150

Set value

Function to set the wait time for a receive using the policy.

amPolSetWaitTime	page 150

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the policy.

amPolClearErrorCodes	page 149
amPolGetLastError	page 149

High-level functions

Each high-level function described in Chapter 3, "The C high-level interface" on page 43 calls a number of the object interface functions, as shown in the following table.

Table 2. Object interface calls used by the high-level functions

High-level function	Equivalent object interface calls
amBackout	amSesCreatePolicy / amSesGetPolicyHandle amSesRollback
amBegin	amSesCreatePolicy / amSesGetPolicyHandle amSesBegin
amBrowseMsg	amSesCreateReceiver / amSesGetReceiverHandle amSesCreatPolicy / amSesGetPolicyHandle amSesCreateMessage / amSesGetMessageHandle amRcvBrowseSelect
amCommit	amSesCreatePolicy / amSesGetPolicyHandle amSesCommit
amInitialize	amSesCreate amSesOpen
amTerminate	amSesClose amSesDelete
amSendMsg amSendRequest amSendResponse	amSesCreateSender / amSesGetSenderHandle amSesCreatePolicy / amSesGetPolicyHandle amSesCreateMessage / amSesGetMessageHandle amSndSend
amReceiveMsg amReceiveRequest	amSesCreateReceiver / amSesGetReceiverHandle amSesCreatePolicy / amSesGetPolicyHandle amSesCreateMessage / amSesGetMessageHandle amRcvReceive
amSendFile	amSesCreateSender / amSesGetSenderHandle amSesCreatePolicy / amSesGetPolicyHandle amSesCreateMessage / amSesGetMessageHandle amSndSendFile
amReceiveFile	amSesCreateReceiver / amSesGetReceiverHandle amSesCreatePolicy / amSesGetPolicyHandle amSesCreateMessage / amSesGetMessageHandle amRcvReceiveFile
amPublish	amSesCreatePublisher / amSesGetPublisherHandle amSesCreatePolicy / amSesGetPolicyHandle amSesCreateMessage / amSesGetMessageHandle amPubPublish
amSubscribe	amSesCreateSubscriber / amSesGetSubscribeHandle amSesCreatePolicy / amSesGetPolicyHandle amSesCreateMessage / amSesGetMessageHandle amSubSubscribe
amUnsubscribe	amSesCreateSubscriber / amSesGetSubscribeHandle amSesCreatePolicy / amSesGetPolicyHandle amSesCreateMessage / amSesGetMessageHandle amSubUnsubscribe
amReceivePublication	amSesCreateSubscriber / amSesGetSubscribeHandle amSesCreatePolicy / amSesGetPolicyHandle amSesCreateMessage / amSesGetMessageHandle amSubReceive

If an object already exists, the appropriate call to get its handle is used instead of calling the create function again. For example, if the message object exists, **amSesGetMessageHandle** is used instead of **amSesCreateMessage**

Chapter 5. C object interface reference

In the following sections the C object interface functions are listed by the object they refer to:

Session	page 84
Message	page 96
Sender	page 117
Receiver	page 122
Distribution list	page 133
Publisher	page 138
Subscriber	page 142
Policy	page 149

Within each section the functions are listed in alphabetical order.

Note that all functions return a completion code (`pCompCode`) and a reason code (`pReason`). The completion code can take one of the following values:

AMCC_OK	Function completed successfully
AMCC_WARNING	Function completed with a warning
AMCC_FAILED	An error occurred during processing

If the completion code returns warning or failed, the reason code identifies the reason for the error or warning (see Appendix A, “Reason codes and LDAP error codes” on page 537).

You can specify the completion code and reason code as null pointers when the function is called, in which case the value is not returned.

Most functions return AMBOOL. They return a value of AMB_TRUE if the function completed successfully, otherwise AMB_FALSE. Functions that do not return AMBOOL return a handle as specified in the following sections.

Most functions require a handle to the object they reference. If this handle is not valid, the results are unpredictable.

Session interface functions

A *session* object provides the scope for a unit of work and creates and manages all other objects, including at least one connection object. Each (WebSphere MQ) connection object encapsulates a single WebSphere MQ queue manager connection. The session object definition specifying the required queue manager connection can be provided by a repository policy definition and the local host file, or the local host file only which by default will name a single local queue manager with no repository. The session, when deleted, is responsible for releasing memory by closing and deleting all other objects that it manages.

Note that you should not mix WebSphere MQ MQCONN or MQDISC requests on the same thread as AMI calls, otherwise premature disconnection might occur.

amSesBegin

Begins a unit of work, allowing an AMI application to take advantage of the resource coordination provided in WebSphere MQ. The unit of work can subsequently be committed by **amSesCommit**, or backed out by **amSesRollback**. It should be used only when WebSphere MQ is the transaction coordinator. If an external transaction coordinator (for example, CICS or Tuxedo) is being used, the API of the external coordinator should be used instead.

```
AMBOOL amSesBegin(
  AMHSES    hSess,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesClearErrorCodes

Clears the error codes in the session object.

```
AMBOOL amSesClearErrorCodes(
  AMHSES    hSess,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesClose

Closes the session object and all open objects owned by the session, and disconnects from the underlying message transport (WebSphere MQ).

```
AMBOOL amSesClose(
  AMHSES    hSess,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesCommit

Commits a unit of work that was started by **amSesBegin**, or by sending or receiving a message under syncpoint control as defined in the policy options for the send or receive request.

```
AMBOOL amSesCommit(
  AMHSES    hSess,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesCreate

Creates the session and system default objects. **amSesCreate** returns the handle of the session object (of type AMHSES). This must be specified by other session function calls.

```
AMHSES amSesCreate(
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

name	An optional session name that can be used to identify the application from which a message is sent (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesCreateDistList

Creates a distribution list object. A distribution list handle (of type AMHDST) is returned.

```
AMHDST amSesCreateDistList(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the distribution list (input). This must match the name of a distribution list defined in the repository.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesCreateMessage

Creates a message object. A message handle (of type AMHMSG) is returned.

```
AMHMSG amSesCreateMessage(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the message (input). This can be any name that is meaningful to the application. It is specified so that this message object can be used with the high-level interface.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesCreatePolicy

Creates a policy object. A policy handle (of type AMHPOL) is returned.

```
AMHPOL amSesCreatePolicy(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the policy (input). If it matches a policy defined in the repository, the policy will be created using the repository definition, otherwise it will be created with default values. If a repository is being used and the named policy is not found in the repository, a completion code of AMCC_WARNING is returned with a reason code of AMRC_POLICY_NOT_IN_REPOS.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesCreatePublisher

Creates a publisher object. A publisher handle (of type AMHPUB) is returned.

```
AMHPUB amSesCreatePublisher(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the publisher (input). If it matches a publisher defined in the repository, the publisher will be created using the repository definition, otherwise it will be created with default values (that is, with a sender service name that matches the publisher name). If a repository is being used and the named publisher is not found in the repository, a completion code of AMCC_WARNING is returned with a reason code of AMRC_PUBLISHER_NOT_IN_REPOS.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesCreateReceiver

Creates a receiver service object. A receiver handle (of type AMHRCV) is returned.

```
AMHRCV amSesCreateReceiver(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the receiver service (input). If it matches a receiver defined in the repository, the receiver will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the receiver name). If a repository is being used and the named receiver is not found in the repository, a completion code of AMCC_WARNING is returned with a reason code of AMRC_RECEIVER_NOT_IN_REPOS.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesCreateSender

Creates a sender service object. A sender handle (of type AMHSND) is returned.

```
AMHSND amSesCreateSender(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the sender service (input). If it matches a sender defined in the repository, the sender will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the sender name). If a repository is being used and the named sender is not found in the repository, a completion code of AMCC_WARNING is returned with a reason code of AMRC_SENDER_NOT_IN_REPOS.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesCreateSubscriber

Creates a subscriber object. A subscriber handle (of type AMHSUB) is returned.

```
AMHSUB amSesCreateSubscriber(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the subscriber (input). If it matches a subscriber defined in the repository, the subscriber will be created using the repository definition, otherwise it will be created with default values (that is, with a sender service name that matches the subscriber name, and a receiver service name that is the same with the addition of the suffix '.RECEIVER'). If a repository is being used and the named subscriber is not found in the repository, a completion code of AMCC_WARNING is returned with a reason code of AMRC_SUBSCRIBER_NOT_IN_REPOS.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesDelete

Deletes the session object. Performs an implicit close if the session is open. This closes and deletes the session and all objects owned by it.

```
AMBOOL amSesDelete(
  PAMHSES   phSess,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

phSess	A *pointer* to the session handle returned by **amSesCreate** (input/output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesDeleteDistList

Deletes a distribution list object, and performs an implicit close if the distribution list is open.

```
AMBOOL amSesDeleteDistList(
  AMHSES    hSess,
  PAMHDST   phDistList,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
phDistList	A *pointer* to the distribution list handle (input/output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesDeleteMessage

Deletes a message object.

```
AMBOOL amSesDeleteMessage(
  AMHSES    hSess,
  PAMHMSG   phMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
phMsg	A *pointer* to the message handle (input/output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesDeletePolicy

Deletes a policy object.

```
AMBOOL amSesDeletePolicy(
  AMHSES    hSess,
  PAMHPOL   phPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
phPolicy	A *pointer* to the policy handle (input/output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesDeletePublisher

Deletes a publisher object, and performs an implicit close if the publisher is open.

```
AMBOOL amSesDeletePublisher(
  AMHSES    hSess,
  PAMHPUB   phPub,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
phPub	A *pointer* to the publisher handle (input/output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesDeleteReceiver

Deletes a receiver object, and performs an implicit close if the receiver is open.

```
AMBOOL amSesDeleteReceiver(
  AMHSES    hSess,
  PAMHRCV   phReceiver,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
phReceiver	A *pointer* to the receiver service handle (input/output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesDeleteSender

Deletes a sender object, and performs an implicit close if the sender is open.

```
AMBOOL amSesDeleteSender(
  AMHSES    hSess,
  PAMHSND   phSender,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
phSender	A *pointer* to the sender service handle (input/output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesDeleteSubscriber

Deletes a subscriber object, and performs an implicit close if the subscriber is open.

```
AMBOOL amSesDeleteSubscriber(
  AMHSES    hSess,
  PAMHSUB   phSub,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
phSub	A *pointer* to the subscriber handle (input/output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesGetDistListHandle

Returns the handle of the distribution list object (of type AMHDST) with the specified name.

```
AMHDST amSesGetDistListHandle(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by the **amSesCreate** function (input).
name	The name of the distribution list (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesGetLastError

Gets the information (completion and reason codes) from the last error for the session.

```
AMBOOL amSesGetLastError(
  AMHSES    hSess,
  AMLONG    buffLen,
  PAMLONG   pStringLen,
  AMSTR     pErrorText,
  PAMLONG   pReason2,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
buffLen	Reserved, must be zero (input).
pStringLen	Reserved, must be NULL (input).
pErrorText	Reserved, must be NULL (input).
pReason2	A secondary reason code (output). Not returned if specified as NULL. If `pReason` indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, `pReason2` gives a WebSphere MQ reason code.
pCompCode	Completion code (output). Not returned if specified as NULL.
pReason	Reason code (output). Not returned if specified as NULL. A value of AMRC_SESSION_HANDLE_ERR indicates that the **amSesGetLastError** function call has itself detected an error and failed.

amSesGetMessageHandle

Returns the handle of the message object (of type AMHMSG) with the specified name.

```
AMHMSG amSesGetMessageHandle(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the message (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesGetPolicyHandle

Returns the handle of the policy object (of type AMHPOL) with the specified name.

```
AMHPOL amSesGetPolicyHandle(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the policy (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesGetPublisherHandle

Returns the handle of the publisher object (of type AMHPUB) with the specified name.

```
AMHPUB amSesGetPublisherHandle(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the publisher (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesGetReceiverHandle

Returns the handle of the receiver service object (of type AMHRCV) with the specified name.

```
AMHRCV amSesGetReceiverHandle(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the receiver service (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesGetSenderHandle

Returns the handle of the sender service object (of type AMHSND) with the specified name.

```
AMHSND amSesGetSenderHandle(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the sender service (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesGetSubscriberHandle

Returns the handle of the subscriber object (of type AMHSUB) with the specified name.

```
AMHSUB amSesGetSubscriberHandle(
  AMHSES    hSess,
  AMSTR     name,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
name	The name of the subscriber (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesOpen

Opens the session object using the specified policy options. The policy, together with the local host file, provides the connection definition that enables the connection object to be created. The specified library is loaded and initialized. If the policy connection type is specified as AUTO and the WebSphere MQ local queue manager library cannot be loaded, the WebSphere MQ client library is loaded. (On z/OS, client connections are not supported so applications must use a local queue manager.) The connection to the underlying message transport (WebSphere MQ) is then opened.

```
AMBOOL amSesOpen(
  AMHSES    hSess,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSess	The session handle returned by **amSesCreate** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSesRollback

Rolls back a unit of work.

```
AMBOOL amSesRollback(
  AMHSES    hSess,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

`hSess`	The session handle returned by **amSesCreate** (input).
`hPolicy`	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
`pCompCode`	Completion code (output).
`pReason`	Reason code (output).

Message interface functions

A *message* object encapsulates a WebSphere MQ message descriptor (MQMD), and name/value elements such as the topic data for publish/subscribe messages. It can also contain the message data, or this can be passed as a separate parameter.

A name/value element in a message object is held in an AMELEM structure. See "Using name/value elements" on page 24 for details.

The initial state of the message object is:

CCSID	default queue manager CCSID
correlationId	all zeros
dataLength	zero
dataOffset	zero
elementCount	zero
encoding	AMENC_NATIVE
format	AMFMT_STRING
groupStatus	AMGRP_MSG_NOT_IN_GROUP
topicCount	zero

When a message object is used to send a message, it will not normally be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see **amMsgReset** on page 109) and rebuild it each time.

Note that the following calls are valid only after a session has been opened with an **amSesOpen** call or after you have explicitly set the element CCSID with an **amMsgSetElementCCSID** call:

amMsgAddElement	page 97
amMsgDeleteElement	page 98
amMsgGetElement	page 101
amMsgGetElementCount	page 102
amMsgDeleteNamedElement	page 99
amMsgGetNamedElement	page 106
amMsgGetNamedElementCount	page 107
amMsgAddTopic	page 98
amMsgDeleteTopic	page 100
amMsgGetTopic	page 108
amMsgGetTopicCount	page 109

amMsgAddElement

Adds a name/value element to a message (such as a publish/subscribe message).

```
AMBOOL amMsgAddElement(
  AMHMSG    hMsg,
  PAMELEM   pElem,
  AMLONG    options,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
pElem	A pointer to an AMELEM element structure, which specifies the element to be added (input). It will not replace an existing element with the same name.
options	A reserved field, which must be set to zero (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgAddFilter

Adds a filter to a subscribe or unsubscribe request message.

```
AMBOOL  amMsgAddFilter(
  AMHMSG    hMsg,
  AMLONG    filterLen,
  AMSTR     pFilter,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hMsg	The message handle returned by **amSesCreateMessage** (input).
filterLen	The length in bytes of the filter (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
pFilter	The filter to be added (input). The syntax of the filter string is described in the *WebSphere MQ Integrator Version 2.0 Programming Guide*.
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgAddTopic

Adds a topic to a publish/subscribe message.

```
AMBOOL amMsgAddTopic(
  AMHMSG    hMsg,
  AMLONG    topicLen,
  AMSTR     pTopic,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
topicLen	The length in bytes of the topic (input). A value of AMLEN_NULL_TERM specifies that the string is NULL terminated.
pTopic	The topic to be added (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgClearErrorCodes

Clears the error codes in the message object.

```
AMBOOL amMsgClearErrorCodes(
  AMHMSG    hMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgDeleteElement

Deletes an element with the specified index from a message (such as a publish/subscribe message). Indexing is within all elements of the message, and might include topics or filters (which are specialized elements).

```
AMBOOL amMsgDeleteElement(
  AMHMSG    hMsg,
  AMLONG    elemIndex,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
elemIndex	The index of the required element in the message, starting from zero (input). On completion, elements with higher `elemIndex` values than that specified will have their index value reduced by one. **amMsgGetElementCount** gets the number of elements in the message.
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgDeleteFilter

Deletes a filter from a subscribe or unsubscribe request message at the specified index. Indexing is within all filters.

```
AMBOOL  amMsgDeleteFilter(
  AMHMSG    hMsg,             /* Message handle */
  AMLONG    filterIndex,      /* Filter index */
  PAMLONG   pCompCode,        /* Completion code */
  PAMLONG   pReason);         /* Reason code qualifying CompCode */
```

Parameters

hMsg	The message handle returned by amSesCreateMessage (input).
filterIndex	The index of the required filter in the message, starting from zero (input). **amMsgGetFilterCount** gets the number of filters in the message.
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgDeleteNamedElement

Deletes a named element from a message, at the specified index. Indexing is within all elements that share the same name.

```
AMBOOL amMsgDeleteNamedElement(
  AMHMSG    hMsg,
  AMLONG    nameIndex,
  AMLONG    nameLen,
  AMSTR     pName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
nameIndex	The index of the required named element in the message (input). Specifying an index of zero deletes the *first* element with the specified name. On completion, elements with higher `nameIndex` values than that specified will have their index value reduced by one. **amMsgGetNamedElementCount** gets the number of elements in the message with the specified name.
nameLen	The length of the element name, in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is NULL terminated.
pName	The name of the element to be deleted (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgDeleteTopic

Deletes a topic from a publish/subscribe message, at the specified index. Indexing is within all topics in the message.

```
AMBOOL amMsgDeleteTopic(
  AMHMSG    hMsg,
  AMLONG    topicIndex,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
topicIndex	The index of the required topic in the message, starting from zero (input). **amMsgGetTopicCount** gets the number of topics in the message.
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetCCSID

Gets the coded character set identifier of the message.

```
AMBOOL amMsgGetCCSID(
  AMHMSG    hMsg,
  PAMLONG   pCCSID,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
pCCSID	The coded character set identifier (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetCorrelId

Gets the correlation identifier of the message.

```
AMBOOL amMsgGetCorrelId(
  AMHMSG    hMsg,
  AMLONG    buffLen,
  PAMLONG   pCorrelIdLen,
  PAMBYTE   pCorrelId,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
buffLen	The length in bytes of a buffer in which the correlation identifier is returned (input).
pCorrelIdLen	The length of the correlation identifier, in bytes (output). If specified as NULL, the length is not returned.
pCorrelId	The correlation identifier (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetDataLength

Gets the length of the message data in the message object.

```
AMBOOL amMsgGetDataLength(
  AMHMSG    hMsg,
  PAMLONG   pLength,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
pLength	The length of the message data, in bytes (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetDataOffset

Gets the current offset in the message data for reading or writing data bytes.

```
AMBOOL amMsgGetDataOffset(
  AMHMSG    hMsg,
  PAMLONG   pOffset,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
pOffset	The byte offset in the message data (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetElement

Gets an element from a message (such as a publish/subscribe message).

```
AMBOOL amMsgGetElement(
  AMHMSG    hMsg,
  AMLONG    elemIndex,
  PAMELEM   pElem,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
elemIndex	The index of the required element in the message, starting from zero (input). **amMsgGetElementCount** gets the number of elements in the message.
pElem	The selected element in the message (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetElementCCSID

Gets the message element CCSID. This is the coded character set identifier used for passing message element data (including topic and filter data) to or from an application.

```
AMBOOL amMsgGetElementCCSID(
  AMHMSG    hMsg,
  PAMLONG   pElementCCSID,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
pElementCCSID	The element coded character set identifier (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetElementCount

Gets the total number of elements in a message (such as a publish/subscribe message).

```
AMBOOL amMsgGetElementCount(
  AMHMSG    hMsg,
  PAMLONG   pCount,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
pCount	The number of elements in the message (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetEncoding

Gets the value used to encode numeric data types for the message.

```
AMBOOL amMsgGetEncoding(
  AMHMSG    hMsg,
  PAMLONG   pEncoding,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
pEncoding	The encoding of the message (output). The following values can be returned: AMENC_NATIVE AMENC_NORMAL AMENC_NORMAL_FLOAT_390 AMENC_REVERSED AMENC_REVERSED_FLOAT_390 AMENC_UNDEFINED
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetFilter

Get a filter from a publish/subscribe message, at the specified index. Indexing is within all filters.

```
AMBOOL  amMsgGetFilter(
  AMHMSG    hMsg,
  AMLONG    filterIndex,
  AMLONG    buffLen,
  PAMLONG   pFilterLen,
  AMSTR     pFilter,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hMsg	The message handle returned by amSesCreateMessage (input).
filterIndex	The index of the required filter in the message (input). Specifying an index of zero returns the first filter. **amMsgGetFilterCount** gets the number of filters in a message.
buffLen	The length in bytes of a buffer in which the filter is returned (input).
pFilterLen	The length of the filter, in bytes (output).
pFilter	The filter (output)
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetFilterCount

Gets the total number of filters in a publish/subscribe message.

```
AMBOOL  amMsgGetFilterCount(
  AMHMSG    hMsg,
  PAMLONG   pCount,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

Parameters

hMsg	The message handle returned by amSesCreateMessage (input).
pCount	The number of filters (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetFormat

Gets the format of the message.

```
AMBOOL amMsgGetFormat(
  AMHMSG    hMsg,
  AMLONG    buffLen,
  PAMLONG   pFormatLen,
  AMSTR     pFormat,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
buffLen	The length in bytes of a buffer in which the format is returned (input).
pFormatLen	The length of the format, in bytes (output). If specified as NULL, the length is not returned.
pFormat	The format of the message (output). The values that can be returned include the following: AMFMT_NONE AMFMT_STRING AMFMT_RF_HEADER
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetGroupStatus

Gets the group status of the message. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group.

```
AMBOOL amMsgGetGroupStatus(
  AMHMSG    hMsg,
  PAMLONG   pStatus,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
pStatus	The group status (output). It can take one of the following values: AMGRP_MSG_NOT_IN_GROUP AMGRP_FIRST_MSG_IN_GROUP AMGRP_MIDDLE_MSG_IN_GROUP AMGRP_LAST_MSG_IN_GROUP AMGRP_ONLY_MSG_IN_GROUP Alternatively, bitwise tests can be performed using the constants: AMGF_IN_GROUP AMGF_FIRST AMGF_LAST
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetLastError

Gets the information (completion and reason codes) from the last error for the message object.

```
AMBOOL amMsgGetLastError(
  AMHMSG    hMsg,
  AMLONG    buffLen,
  PAMLONG   pStringLen,
  AMSTR     pErrorText,
  PAMLONG   pReason2,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

`hMsg`	The message handle returned by **amSesCreateMessage** (input).
`buffLen`	Reserved, must be zero (input).
`pStringLen`	Reserved, must be NULL (input).
`pErrorText`	Reserved, must be NULL (input).
`pReason2`	A secondary reason code (output). Not returned if specified as NULL. If `pReason` indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, `pReason2` gives a WebSphere MQ reason code.
`pCompCode`	Completion code (output). Not returned if specified as NULL.
`pReason`	Reason code (output). Not returned if specified as NULL. A value of AMRC_MSG_HANDLE_ERR indicates that the **amMsgGetLastError** function call has itself detected an error and failed.

amMsgGetMsgId

Gets the message identifier.

```
AMBOOL amMsgGetMsgId(
  AMHMSG    hMsg,
  AMLONG    buffLen,
  PAMLONG   pMsgIdLen,
  PAMBYTE   pMsgId,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

`hMsg`	The message handle returned by **amSesCreateMessage** (input).
`buffLen`	The length in bytes of a buffer in which the message identifier is returned (input).
`pMsgIdLen`	The length of the message identifier, in bytes (output). If specified as NULL, the length is not returned.
`pMsgId`	The message identifier (output).
`pCompCode`	Completion code (output).
`pReason`	Reason code (output).

amMsgGetName

Gets the name of the message object.

```
AMBOOL amMsgGetName(
  AMHMSG    hMsg,
  AMLONG    buffLen,
  PAMLONG   pNameLen,
  AMSTR     pName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
buffLen	The length in bytes of a buffer into which the name is put (input). If specified as zero, only the name length is returned.
pNameLen	The length of the name, in bytes (output). If specified as NULL, only the name is returned.
pName	The message object name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetNamedElement

Gets a named element from a message (such as a publish/subscribe message).

```
AMBOOL amMsgGetNamedElement(
  AMHMSG    hMsg,
  AMLONG    nameIndex,
  AMLONG    nameLen,
  AMSTR     pName,
  PAMELEM   pElem,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
nameIndex	The index of the required named element in the message (input). Specifying an index of zero returns the first element with the specified name. **amMsgGetNamedElementCount** gets the number of elements in the message with the specified name.
nameLen	The length of the element name, in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
pName	The element name (input).
pElem	The selected named element in the message (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetNamedElementCount

Gets the number of elements in a message with a specified name.

```
AMBOOL amMsgGetNamedElementCount(
  AMHMSG    hMsg,
  AMLONG    nameLen,
  AMSTR     pName,
  PAMLONG   pCount,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
nameLen	The length of the element name, in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
pName	The specified element name (input).
pCount	The number of elements in the message with the specified name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetReportCode

Gets the feedback code from a message of type AMMT_REPORT. If the message type is not AMMT_REPORT, error code AMRC_MSG_TYPE_NOT_REPORT will be returned.

```
AMBOOL amMsgGetReportCode(
  AMHMSG    hMsg,
  PAMLONG   pCode,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg The message handle returned by **amSesCreateMessage** (input).

PCode The feedback code (output). In addition to application defined values, the following can be returned:

```
AMFB_CODE_EXPIRATION
AMFB_CODE_COA
AMFB_CODE_COD
AMFB_NONE
```

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetTopic

Gets a topic from a publish/subscribe message, at the specified index. Indexing is within all topics.

```
AMBOOL amMsgGetTopic(
  AMHMSG    hMsg,
  AMLONG    topicIndex,
  AMLONG    buffLen,
  PAMLONG   pTopicLen,
  AMSTR     pTopic,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
topicIndex	The index of the required topic in the message (input). Specifying an index of zero returns the first topic. **amMsgGetTopicCount** gets the number of topics in the message.
buffLen	The length in bytes of a buffer in which the topic is returned (input). If `buffLen` is specified as zero, only the topic length is returned (in `pTopicLen`), not the topic itself.
pTopicLen	The length of the topic, in bytes (output).
pTopic	The topic (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetType

Gets the message type from a message.

```
AMBOOL amMsgGetType(
  AMHMSG    hMsg,
  PAMLONG   pType,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).

PType	The message type (output). The following values can be returned:
	`AMMT_DATAGRAM`
	`AMMT_REQUEST`
	`AMMT_REPLY`
	`AMMT_REPORT`

pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetTopicCount

Gets the total number of topics in a publish/subscribe message.

```
AMBOOL amMsgGetTopicCount(
  AMHMSG    hMsg,
  PAMLONG   pCount,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
pCount	The number of topics (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgReadBytes

Reads up to the specified number of data bytes from the message object, starting at the current data offset (which must be positioned before the end of the data for the read operation to be successful). Use **amMsgSetDataOffset** to set the data offset. **amMsgReadBytes** will advance the data offset by the number of bytes read, leaving the offset immediately after the last byte read.

```
AMBOOL amMsgReadBytes(
  AMHMSG    hMsg,
  AMLONG    readLen,
  PAMLONG   pBytesRead,
  PAMBYTE   pData,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
readLen	The maximum number of bytes to be read (input). The data buffer specified by `pData` must be at least this size. The number of bytes returned is the minimum of `readLen` and the number of bytes between the data offset and the end of the data.
pBytesRead	The number of bytes read (output). If specified as NULL, the number is not returned.
pData	The read data (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgReset

Resets the message object its initial state (see page 96).

```
AMBOOL amMsgReset(
  AMHMSG    hMsg,
  AMLONG    options,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
options	A reserved field that must be specified as zero (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgSetCCSID

Sets the coded character set identifier of the message.

```
AMBOOL amMsgSetCCSID(
  AMHMSG    hMsg,
  AMLONG    CCSID,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
CCSID	The coded character set identifier (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgSetCorrelId

Sets the correlation identifier of the message.

```
AMBOOL amMsgSetCorrelId(
  AMHMSG    hMsg,
  AMLONG    correlIdLen,
  PAMBYTE   pCorrelId,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
correlIdLen	The length of the correlation identifier, in bytes (input). A value of AMLEN_NULL_TERM specifies that the correlation identifier is a string that is null terminated.
pCorrelId	The correlation identifier (input). Specify as NULL (with a `correlIdLen` of 0L) to set the correlation identifier to NULL.
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgSetDataOffset

Sets the data offset for reading or writing byte data. If the data offset is greater than the current data length, it is valid to write data into the message at that offset, but an attempt to read data will result in an error. See “amMsgReadBytes” on page 109 and “amMsgWriteBytes” on page 114.

```
AMBOOL amMsgSetDataOffset(
  AMHMSG    hMsg,
  AMLONG    offset,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
offset	The offset in bytes (input). Set an offset of zero to read or write from the start of the data.
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgSetElementCCSID

This specifies the character set to be used for subsequent element message data (including topic and filter data) passed to or returned from the application. Existing elements in the message are unmodified (but will be returned in this character set). The default value of element CCSID is the queue manager CCSID.

```
AMBOOL amMsgSetElementCCSID(
  AMHMSG    hMsg,
  AMLONG    elementCCSID,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
elementCCSID	The element coded character set identifier (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgSetEncoding

Sets the encoding of the data in the message.

```
AMBOOL amMsgSetEncoding(
  AMHMSG    hMsg,
  AMLONG    encoding,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
encoding	The encoding of the message (input). It can take one of the following values: AMENC_NATIVE AMENC_NORMAL AMENC_NORMAL_FLOAT_390 AMENC_REVERSED AMENC_REVERSED_FLOAT_390 AMENC_UNDEFINED
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgSetFormat

Sets the format of the message.

```
AMBOOL amMsgSetFormat(
  AMHMSG    hMsg,
  AMLONG    formatLen,
  AMSTR     pFormat,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
formatLen	The length of the format, in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is NULL terminated.

pFormat	The format of the message (input). It can take one of the following values, or an application defined string:
	`AMFMT_NONE`
	`AMFMT_STRING`
	`AMFMT_RF_HEADER`
	If set to AMFMT_NONE, the default format for the sender will be used (if available). Specify as NULL (with a `formatLen` of 0) to set the format to NULL.

pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgSetGroupStatus

Sets the group status of the message. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group. Once you start sending messages in a group, you must complete the group before sending any messages that are not in the group.

If you specify AMGRP_MIDDLE_MSG_IN_GROUP or AMGRP_LAST_MSG_IN_GROUP without specifying AMGRP_FIRST_MSG_IN_GROUP, the behavior is the same as for AMGRP_FIRST_MSG_IN_GROUP and AMGRP_ONLY_MSG_IN_GROUP respectively.

If you specify AMGRP_FIRST_MSG_IN_GROUP out of sequence, the behavior is the same as for AMGRP_MIDDLE_MSG_IN_GROUP.

```
AMBOOL amMsgSetGroupStatus(
  AMHMSG    hMsg,
  AMLONG    status,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).

status	The group status (input). It can take one of the following values:
	`AMGRP_MSG_NOT_IN_GROUP`
	`AMGRP_FIRST_MSG_IN_GROUP`
	`AMGRP_MIDDLE_MSG_IN_GROUP`
	`AMGRP_LAST_MSG_IN_GROUP`
	`AMGRP_ONLY_MSG_IN_GROUP`

pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgSetReportCode

Sets the feedback code type for a message. This is meaningful only for a message of type AMMT_REPORT.

```
AMBOOL amMsgSetReportCode(
  AMHMSG    hMsg,
  AMLONG    code,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
code	The feedback (or report code) value (input). In addition to application defined values, the following can be set: AMFB_CODE_EXPIRATION AMFB_CODE_COA AMFB_CODE_COD AMFB_NONE
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgSetType

Sets the message type. If a response message is requested (on a send) with a publish, subscribe, or unsubscribe request, the value specified here is ignored and message type AMMT_REQUEST is used. If the value specified here is AMMT_DATAGRAM, this is overridden when requesting (on a send) or sending a response message (by AMMT_REQUEST and AMMT_RESPONSE, respectively). Otherwise, the value specified here sets the message type for a message when it is sent.

```
AMBOOL amMsgSetType(
  AMHMSG    hMsg,
  AMLONG    type,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
type	The message type (input). It can take one of the following values: AMMT_DATAGRAM AMMT_REQUEST AMMT_REPLY AMMT_REPORT
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgWriteBytes

Writes the specified number of data bytes into the message object, starting at the current data offset. See “amMsgSetDataOffset” on page 110.

If the data offset is not at the end of the data, existing data is overwritten. If the data offset is set beyond the current data length, the message data between the data length and the data offset is undefined. This feature enables applications to construct messages in a non-sequential manner, but care must be taken to ensure that a message is completely filled with data before it is sent.

amMsgWriteBytes will advance the data offset by the number of bytes written, leaving it immediately after the last byte written.

```
AMBOOL amMsgWriteBytes(
  AMHMSG    hMsg,
  AMLONG    writeLen,
  PAMBYTE   pByteData,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
writeLen	The number of bytes to be written (input).
pByteData	The data bytes (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

Message interface helper macros

The following helper macros are provided for manipulation of the name/value elements in a message object. Additional helper macros can be written as required.

AmMsgAddStreamName

Adds a name/value element for the publish/subscribe stream name.

```
AmMsgAddStreamName(
  AMHMSG    hMsg,
  AMLONG    streamNameLen,
  AMSTR     pStreamName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
streamNameLen	The length of the stream name, in bytes (input).
pStreamName	The stream name (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

AmMsgGetPubTimeStamp

Gets the publication time stamp name/value element.

```
AmMsgGetPubTimeStamp(
  AMHMSG    hMsg,
  AMLONG    buffLen,
  PAMLONG   pTimestampLen,
  AMSTR     pTimestamp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
buffLen	The length in bytes of a buffer in which the publication time stamp is returned (input). Specify as zero to return only the length.
pTimestampLen	The length of the publication time stamp, in bytes (output). If specified as NULL, the length is not returned.
pTimestamp	The publication time stamp (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

AmMsgGetStreamName

Gets the name/value element for the publish/subscribe stream name.

```
AmMsgGetStreamName(
  AMHMSG    hMsg,
  AMLONG    buffLen,
  PAMLONG   pStreamNameLen,
  AMSTR     pStreamName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle returned by **amSesCreateMessage** (input).
buffLen	The length in bytes of a buffer in which the stream name is returned (input). Specify as zero to return only the length.
pStreamNameLen	The length of the stream name, in bytes (output). If specified as NULL, the length is not returned.
pStreamName	The stream name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

Sender interface functions

A *sender* object encapsulates a WebSphere MQ object descriptor (MQOD) structure. This represents a WebSphere MQ queue on a local or remote queue manager. An open sender service is always associated with an open connection object (such as a queue manager connection). Support is also included for dynamic sender services (those that encapsulate model queues). The required sender service object definitions can be provided from a repository, or created without a repository definition by defaulting to the existing queue objects on the local queue manager.

The high-level functions **amSendMsg**, **amSendRequest** and **amSendResponse** call these interface functions as required to open the sender service and send a message. Additional calls are provided here to give the application program extra functionality.

A sender service object must be created before it can be opened. This is done implicitly using the high-level functions, or the **amSesCreateSender** session interface functions.

A *response* sender service is a special type of sender service used for sending a response to a request message. It must be created using the default definition, and not a definition stored in a repository (see "Services, policies, and policy handlers" on page 491). Once created, it must not be opened until used in its correct context as a response sender when receiving a request message with **amRcvReceive** or **amReceiveRequest**. When opened, its queue and queue manager properties are modified to reflect the `ReplyTo` destination specified in the message being received. When first used in this context, the sender service becomes a response sender service.

amSndClearErrorCodes

Clears the error codes in the sender object.

```
AMBOOL amSndClearErrorCodes(
  AMHSND    hSender,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSender	The sender handle returned by **amSesCreateSender** (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSndClose

Closes the sender service.

```
AMBOOL amSndClose(
  AMHSND    hSender,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSender	The sender handle returned by **amSesCreateSender** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSndGetCCSID

Gets the coded character set identifier of the sender service. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the sender must perform CCSID conversion of the message before it is sent.

```
AMBOOL amSndGetCCSID(
  AMHSND    hSender,
  PAMLONG   pCCSID,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSender	The sender handle returned by **amSesCreateSender** (input).
pCCSID	The coded character set identifier (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSndGetEncoding

Gets the value used to encode numeric data types for the sender service. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the sender must convert the encoding of the message before it is sent.

```
AMBOOL amSndGetEncoding(
  AMHSND    hSender,
  PAMLONG   pEncoding,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSender	The sender handle returned by **amSesCreateSender** (input).
pEncoding	The encoding (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSndGetLastError

Gets the information (completion and reason codes) from the last error for the sender object.

```
AMBOOL amSndGetLastError(
  AMHSND    hSender,
  AMLONG    buffLen,
  PAMLONG   pStringLen,
  AMSTR     pErrorText,
  PAMLONG   pReason2,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSender	The sender handle returned by **amSesCreateSender** (input).
buffLen	Reserved, must be zero (input).
pStringLen	Reserved, must be NULL (input).
pErrorText	Reserved, must be NULL (input).
pReason2	A secondary reason code (output). Not returned if specified as NULL. If `pReason` indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, `pReason2` gives a WebSphere MQ reason code.

pCompCode	Completion code (output). Not returned if specified as NULL.
pReason	Reason code (output). Not returned if specified as NULL. A value of AMRC_SERVICE_HANDLE_ERR indicates that the **amSndGetLastError** function call has itself detected an error and failed.

amSndGetName

Gets the name of the sender service.

```
AMBOOL amSndGetName(
  AMHSND    hSender,
  AMLONG    buffLen,
  PAMLONG   pNameLen,
  AMSTR     pName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSender	The sender handle returned by **amSesCreateSender** (input).
buffLen	The length in bytes of a buffer in which the name is returned (input). If specified as zero, only the name length is returned.
pNameLen	The length of the name, in bytes (output). If specified as NULL, only the name is returned.
pName	The name of the sender service (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSndOpen

Opens the sender service.

```
AMBOOL amSndOpen(
  AMHSND    hSender,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSender	The sender handle returned by **amSesCreateSender** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSndSend

Sends a message to the destination specified by the sender service. If the sender service is not open, it will be opened (if this action is specified in the policy options).

The message data can be passed in the message object, or as a separate parameter (this means that the data does not have to be copied into the message object before sending the message, which might improve performance, especially if the message data is large).

```
AMBOOL amSndSend(
  AMHSND    hSender,
  AMHPOL    hPolicy,
  AMHRCV    hReceiver,
  AMHMSG    hRcvMsg,
  AMLONG    dataLen,
  PAMBYTE   pData,
  AMHMSG    hSndMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSender	The sender handle returned by **amSesCreateSender** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
hReceiver	The handle of the receiver service to which the response to this message should be sent, if the message being sent is a request message (input). Specify as AMH_NULL_HANDLE if no response is required.
hRcvMsg	The handle of a received message that is being responded to, if this is a response message (input). Specify as AMH_NULL_HANDLE if this is not a response message.
dataLen	The length of the message data, in bytes (input). If specified as zero, any message data will be passed in the message object (`hSndMsg`).
pData	The message data, if `dataLen` is non-zero (input).
hSndMsg	The handle of a message object that specifies the properties of the message being sent (input). If `dataLen` is zero, it can also contain the message data. If specified as AMH_NULL_HANDLE, the default message object (constant: AMSD_SND_MSG_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSndSendFile

Sends data from a file.The file data can be received as normal message data by a target application using **amRcvReceive** or used to reconstruct the file with **amRcvReceiveFile**.

```
AMBOOL amSndSendFile(
  AMHSND    hSender,
  AMHPOL    hPolicy,
  AMLONG    options,
  AMLONG    directoryLen,
  AMSTR     directory,
  AMLONG    fileNameLen,
  AMSTR     fileName,
  AMHMSG    hSndMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSender	The sender handle returned by **amSesCreateSender** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
options	A reserved field that must be specified as zero.
directoryLen	A reserved field that must be specified as zero (input).
directory	A reserved field that must be specified as NULL (input).
fileNameLen	The length of the file name in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
fileName	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the filename will travel with the message for use with a receive file call (see "amRcvReceiveFile" on page 131 for more details). Note that the filename sent will exactly match the supplied filename; it will not be converted or expanded in any way.
hSndMsg	The handle of the message object to use to send the file (input). This can be used to specify the Correlation ID for example. If specified as AMH_NULL_HANDLE, the system default send message (constant: AMSD_SND_MSG_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

If, in your application, you have previously used a message object, referenced by either handle or name, to send or receive data (including AMI elements or topics), you will need to explicitly call amMsgReset before re-using the object for sending a file. This applies even if you use the system default object handle (constant: AMSD_SND_MSG_HANDLE).

Receiver interface functions

A *receiver* object encapsulates a WebSphere MQ object descriptor (MQOD) structure. This represents a local WebSphere MQ queue. An open receiver service is always associated with an open connection object, such as a queue manager connection. Support is also included for dynamic receiver services (that encapsulate model queues). The required receiver service object definitions can be provided from a repository or can be created automatically from the set of existing queue objects available on the local queue manager.

There is a definition type associated with each receiver service:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

A receiver service created from a repository definition will be initially of type AMDT_PREDEFINED or AMDT_DYNAMIC. When opened, its definition type might change from AMDT_DYNAMIC to AMDT_TEMP_DYNAMIC according to the properties of its underlying queue object.

A receiver service created with default values (that is, without a repository definition) will have its definition type set to AMDT_UNDEFINED until it is opened. When opened, this will become AMDT_DYNAMIC, AMDT_TEMP_DYNAMIC, or AMDT_PREDEFINED, according to the properties of its underlying queue object.

amRcvBrowse

Browses a message. See the *WebSphere MQ Application Programming Guide* for a full description of the browse options.

```
AMBOOL amRcvBrowse(
  AMHRCV    hReceiver,
  AMHPOL    hPolicy,
  AMLONG    options,
  AMLONG    buffLen,
  PAMLONG   pDataLen,
  PAMBYTE   pData,
  AMHMSG    hRcvMsg,
  AMHSND    hSender,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver	The receiver handle returned by **amSesCreateReceiver** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
options	Options controlling the browse operation (input). Possible values are:

```
AMBRW_NEXT
AMBRW_FIRST
AMBRW_CURRENT
AMBRW_RECEIVE_CURRENT
AMBRW_DEFAULT          (AMBRW_NEXT)
AMBRW_LOCK_NEXT        (AMBRW_LOCK + AMBRW_NEXT)
AMBRW_LOCK_FIRST       (AMBRW_LOCK + AMBRW_FIRST)
AMBRW_LOCK_CURRENT     (AMBRW_LOCK + AMBRW_CURRENT)
AMBRW_UNLOCK
```

`AMBRW_RECEIVE_CURRENT` is equivalent to **amRcvReceive** for the message under the browse cursor.

Note that a locked message is unlocked by another browse or receive, even though it is not for the same message.

buffLen	The length in bytes of a buffer in which the data is returned (input).
pDataLen	The length of the message data in bytes (output). If specified as NULL, the data length is not returned.
pData	The received message data (output).
hRcvMsg	The handle of the message object for the received message (output).
hSender	The handle of the response sender service that the response message must be sent to, if this is a request message (output). This sender service must be created without a repository definition, and used exclusively for sending a response. Its definition type must be AMDT_UNDEFINED (it will be set to AMDT_RESPONSE by this call). Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

You can return the message data in the message object or in an application buffer.

To return the data in the message object (`hRcvMsg`), set `buffLen` to zero, and set `pData` and `pDataLen` to values that are not NULL.

To return data in an application message buffer:

- set `pData` to the buffer pointer value (that is, not NULL)
- set `buffLen` to the length of the buffer

If the value of `buffLen` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a value for `pDataLen` that is not NULL.

To return only the data length:

- set `pData` to NULL
- set `buffLen` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

amRcvBrowseSelect

Browses a message identified by specifying the Correlation ID from the selection message as a selection criterion. See the *WebSphere MQ Application Programming Guide* for a full description of the browse options.

```
AMBOOL amRcvBrowseSelect(
  AMHRCV    hReceiver,
  AMHPOL    hPolicy,
  AMLONG    options,
  AMHMSG    hSelMsg,
  AMLONG    buffLen,
  PAMLONG   pDataLen,
  PAMBYTE   pData,
  AMHMSG    hRcvMsg,
  AMHSND    hSender,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver	The receiver handle returned by **amSesCreateReceiver** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
options	Options controlling the browse operation (input). Possible values are:

```
AMBRW_NEXT
AMBRW_FIRST
AMBRW_CURRENT
AMBRW_RECEIVE_CURRENT
AMBRW_DEFAULT          (AMBRW_NEXT)
AMBRW_LOCK_NEXT        (AMBRW_LOCK + AMBRW_NEXT)
AMBRW_LOCK_FIRST       (AMBRW_LOCK + AMBRW_FIRST)
AMBRW_LOCK_CURRENT     (AMBRW_LOCK + AMBRW_CURRENT)
AMBRW_UNLOCK
```

`AMBRW_RECEIVE_CURRENT` is equivalent to **amRcvReceive** for the message under the browse cursor.

Note that a locked message is unlocked by another browse or receive, even though it is not for the same message.

hSelMsg	The handle of a selection message object (input). This is used together with the browse options to identify the message to be received (for example, using the Correlation ID). Specify as AMH_NULL_HANDLE to get the next available message. The CCSID, element CCSID, and encoding values from the selection message define the target values for any data conversions. If target conversion values are required without using the Correlation ID for selection, this can be reset (see **amMsgSetCorrelId** on page 110) before invoking the **amRcvBrowseSelect** function.
buffLen	The length in bytes of a buffer in which the data is returned (input).
pDataLen	The length of the message data in bytes (output). If specified as NULL, the data length is not returned.
pData	The received message data (output).
hRcvMsg	The handle of the message object for the received message (output).
hSender	The handle of the response sender service that the response

message must be sent to, if this is a request message (output). This sender service must be created without a repository definition, and used exclusively for sending a response. Its definition type must be AMDT_UNDEFINED (it will be set to AMDT_RESPONSE by this call).

Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.

pCompCode Completion code (output).

pReason Reason code (output).

Usage notes

You can return the message data in the message object or in an application buffer.

To return the data in the message object (`hRcvMsg`), set `buffLen` to zero, and set `pData` and `pDataLen` to values that are not NULL.

To return data in an application message buffer:

- set `pData` to the buffer pointer value (that is, not NULL)
- set `buffLen` to the length of the buffer

If the value of `buffLen` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a value for `pDataLen` that is not NULL.

To return only the data length:

- set `pData` to NULL
- set `buffLen` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

amRcvClearErrorCodes

Clears the error codes in the receiver service object.

```
AMBOOL amRcvClearErrorCodes(
  AMHRCV    hReceiver,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver The receiver handle returned by **amSesCreateReceiver** (input).

pCompCode Completion code (output).

pReason Reason code (output).

C receiver interface

amRcvClose

Closes the receiver service.

```
AMBOOL amRcvClose(
  AMHRCV    hReceiver,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver	The receiver handle returned by **amSesCreateReceiver** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amRcvGetDefnType

Gets the definition type of the receiver service.

```
AMBOOL amRcvGetDefnType(
  AMHRCV    hReceiver,
  PAMLONG   pType,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver	The receiver handle returned by **amSesCreateReceiver** (input).

pType	The definition type (output). It can be one of the following:
	`AMDT_UNDEFINED`
	`AMDT_TEMP_DYNAMIC`
	`AMDT_DYNAMIC`
	`AMDT_PREDEFINED`
	Values other than AMDT_UNDEFINED reflect the properties of the underlying queue object.

pCompCode	Completion code (output).
pReason	Reason code (output).

amRcvGetLastError

Gets the information (completion and reason codes) from the last error for the receiver object.

```
AMBOOL amRcvGetLastError(
  AMHRCV    hReceiver,
  AMLONG    buffLen,
  PAMLONG   pStringLen,
  AMSTR     pErrorText,
  PAMLONG   pReason2,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver	The receiver handle returned by **amSesCreateReceiver** (input).
buffLen	Reserved, must be zero (input).
pStringLen	Reserved, must be NULL (input).
pErrorText	Reserved, must be NULL (input).
pReason2	A secondary reason code (output). Not returned if specified as NULL. If `pReason` indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, `pReason2` gives a WebSphere MQ reason code.
pCompCode	Completion code (output). Not returned if specified as NULL.
pReason	Reason code (output). Not returned if specified as NULL. A value of AMRC_SERVICE_HANDLE_ERR indicates that the **amRcvGetLastError** function call has itself detected an error and failed.

amRcvGetName

Gets the name of the receiver service.

```
AMBOOL amRcvGetName(
  AMHRCV    hReceiver,
  AMLONG    buffLen,
  PAMLONG   pNameLen,
  AMSTR     pName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver	The receiver handle returned by **amSesCreateReceiver** (input).
buffLen	The length in bytes of a buffer into which the name is put (input). Set it to zero to return only the name length.
pNameLen	The length of the name, in bytes (output). Set it to NULL to return only the name.
pName	The name of the receiver service (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amRcvGetQueueName

Gets the queue name of the receiver service. This is used to determine the queue name of a permanent dynamic receiver service, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **amRcvSetQueueName**.)

```
AMBOOL amRcvGetQueueName(
  AMHRCV    hReceiver,
  AMLONG    buffLen,
  PAMLONG   pNameLen,
  AMSTR     pQueueName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver	The receiver handle returned by **amSesCreateReceiver** (input).
buffLen	The length in bytes of a buffer in which the queue name is returned (input).
pNameLen	The length of the queue name, in bytes (output).
pQueueName	The queue name of the receiver service (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amRcvOpen

Opens the receiver service.

```
AMBOOL amRcvOpen(
  AMHRCV    hReceiver,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver	The receiver handle returned by **amSesCreateReceiver** (input).
	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amRcvReceive

Receives a message.

```
AMBOOL amRcvReceive(
  AMHRCV    hReceiver,
  AMHPOL    hPolicy,
  AMHMSG    hSelMsg,
  AMLONG    buffLen,
  PAMLONG   pDataLen,
  PAMBYTE   pData,
  AMHMSG    hRcvMsg,
  AMHSND    hSender,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver	The receiver handle returned by **amSesCreateReceiver** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
hSelMsg	The handle of a selection message object (input). This is used to identify the message to be received (for example, using the correlation ID). Specify as AMH_NULL_HANDLE to get the next available message with no selection. The CCSID, element CCSID, and encoding values from the selection message define the target values for any data conversions. If target conversion values are required without using the Correlation ID for selection, this can be reset (see **amMsgSetCorrelId** on page 84) before invoking the **amRcvReceive** function.
buffLen	The length in bytes of a buffer in which the data is returned (input).
pDataLen	The length of the message data, in bytes (output). If specified as NULL, the data length is not returned.
pData	The received message data (output).
hRcvMsg	The handle of the message object for the received message (output). If specified as AMH_NULL_HANDLE, the default message object (constant: AMSD_RCV_MSG_HANDLE) is used. The message object is reset implicitly before the receive takes place.
hSender	The handle of the response sender service that a response message must be sent to, if this is a request message (output). This sender service must be created without a repository definition, and used exclusively for sending a response. Its definition type must be AMDT_UNDEFINED (it will be set to AMDT_RESPONSE by this call).
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

You can return the message data in the message object or in an application buffer.

To return the data in the message object (`hRcvMsg`), set `buffLen` to zero, and set `pData` and `pDataLen` to values that are not NULL.

To return data in an application message buffer:

- set `pData` to the buffer pointer value (that is, not NULL)

- set `buffLen` to the length of the buffer

If the value of `buffLen` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a value for `pDataLen` that is not NULL.

To return only the data length without removing the message from the queue:
- set `pData` to NULL
- set `buffLen` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

To remove the message from the queue and discard it:
- set `pData` or `pDataLen` to a value that is not NULL
- set `buffLen` to zero
- ensure that Accept Truncated Message in the policy receive attributes is selected

The message will be discarded with an AMRC_MSG_TRUNCATED warning.

If AMRC_RECEIVE_BUFF_LEN_ERR is returned, the message length value is returned in `pDataLen` (if it is not NULL), even though the completion code is MQCC_FAILED.

Note that if `pData` is NULL and `buffLen` is not zero, there is always an AMRC_RECEIVE_BUFF_LEN_ERR error.

amRcvReceiveFile

Receives file message data into a file.

```
AMBOOL amRcvReceiveFile(
  AMHRCV    hReceiver,
  AMHPOL    hPolicy,
  AMHLONG   options,
  AMHMSG    hSelMsg,
  AMLONG    directoryLen,
  AMSTR     directory,
  AMLONG    fileNameLen,
  AMSTR     fileName,
  AMHMSG    hRcvMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hReceiver	The receiver handle returned by **amSesCreateReceiver** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
options	A reserved field that must be specified as zero (input).
hSelMsg	The handle of a selection message object (input). This is used to identify the message to be received (for example, using the correlation ID). Specify as AMH_NULL_HANDLE to get the next available message with no selection. The CCSID, element CCSID, and encoding values from the selection message define the target values for any data conversions. If target conversion values are required without using the Correlation ID for selection, this can be reset (see **amMsgSetCorrelId** on page 110) before invoking the **amRcvReceiveFile** function.
directoryLen	A reserved field that must be specified as zero (input).
directory	A reserved field that must be specified as NULL (input).
fileNameLen	The length of the file name in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated, in which case the AMI will work out the length itself.
fileName	The name of the file into which the transferred data is to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If NULL or a null string is specified, the AMI will use the name of the originating file (including any directory prefix), exactly as it was supplied on the send file call. Note that the original filename may not be appropriate for use by the receiver, either because a pathname included in the filename is not applicable to the receiving system, or because the sending and receiving systems use different filename conventions.
hRcvMsg	The handle of the message object to use to receive the file. This parameter is updated with the message properties, for example the Message ID. If the message is not a file message, `hRcvMsg` receives the message data. If `hRcvMsg` is specified as AMH_NULL_HANDLE, the default message object (constant AMSD_RCV_MSG_HANDLE) is used. The message object is reset implicitly before the receive takes place.
pCompCode	Completion code (output).

`pReason`	Reason code (output).

Usage notes

If `fileName` is blank (indicating that the originating file name specified in the message is to be used), `fileNameLength` should be set to zero.

amRcvSetQueueName

Sets the queue name of the receiver service, when this encapsulates a model queue. This can be used to specify the queue name of a recreated permanent dynamic receiver service, in order to receive messages in a session subsequent to the one in which it was created. (See also **amRcvGetQueueName**.)

```
AMBOOL amRcvSetQueueName(
  AMHRCV    hReceiver,
  AMLONG    nameLen,
  AMSTR     pQueueName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

`hReceiver`	The receiver handle returned by **amSesCreateReceiver** (input).
`nameLen`	The length of the queue name, in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is NULL terminated.
`pQueueName`	The queue name of the receiver service (input).
`pCompCode`	Completion code (output).
`pReason`	Reason code (output).

Distribution list interface functions

A *distribution list* object encapsulates a list of sender objects.

amDstClearErrorCodes

Clears the error codes in the distribution list object.

```
AMBOOL amDstClearErrorCodes(
  AMHDST    hDistList,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hDistList	The distribution list handle returned by **amSesCreateDistList** (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amDstClose

Closes the distribution list.

```
AMBOOL amDstClose(
  AMHDST    hDistList,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hDistList	The distribution list handle returned by **amSesCreateDistList** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

C distribution list interface

amDstGetLastError

Gets the information (completion and reason codes) from the last error in the distribution list object.

```
AMBOOL amDstGetLastError(
  AMHDST    hDistList,
  AMLONG    buffLen,
  PAMLONG   pStringLen,
  AMSTR     pErrorText,
  PAMLONG   pReason2,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hDistList	The distribution list handle returned by **amSesCreateDistList** (input).
buffLen	Reserved, must be zero (input).
pStringLen	Reserved, must be NULL (input).
pErrorText	Reserved, must be NULL (input).
pReason2	A secondary reason code (output). Not returned if specified as NULL. If `pReason` indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, `pReason2` gives a WebSphere MQ reason code.
pCompCode	Completion code (output). Not returned if specified as NULL.
pReason	Reason code (output). Not returned if specified as NULL. A value of AMRC_SERVICE_HANDLE_ERR indicates that the **amDstGetLastError** function call has itself detected an error and failed.

amDstGetName

Gets the name of the distribution list object.

```
AMBOOL amDstGetName(
  AMHDST    hDistList,
  AMLONG    buffLen,
  PAMLONG   pNameLen,
  AMSTR     pName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hDistList	The distribution list handle returned by **amSesCreateDistList** (input).
buffLen	The length in bytes of a buffer into which the name is put (input). Set it to zero to return only the name length.
pNameLen	The length of the name, in bytes (output). Set it to NULL to return only the name.
pName	The distribution list object name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amDstGetSenderCount

Gets a count of the number of sender services in the distribution list.

```
AMBOOL amDstGetSenderCount(
  AMHDST    hDistList,
  PAMLONG   pCount,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hDistList	The distribution list handle returned by **amSesCreateDistList** (input).
pCount	The number of sender services (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amDstGetSenderHandle

Returns the handle (type AMHSND) of a sender service in the distribution list object with the specified index.

```
AMHSND amDstGetSenderHandle(
  AMHDST    hDistList,
  AMLONG    handleIndex,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hDistList	The distribution list handle returned by **amSesCreateDistList** (input).
handleIndex	The index of the required sender service in the distribution list (input). Specify an index of zero to return the first sender service in the list. **amDstGetSenderCount** gets the number of sender services in the distribution list.
pCompCode	Completion code (output).
pReason	Reason code (output).

amDstOpen

Opens the distribution list object for each of the destinations in the distribution list. The completion and reason codes returned by this function call indicate if the open was unsuccessful, partially successful, or completely successful.

```
AMBOOL amDstOpen(
  AMHDST    hDistList,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hDistList	The distribution list handle returned by **amSesCreateDistList** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

C distribution list interface

amDstSend

Sends a message to each sender in the distribution list.

```
AMBOOL amDstSend(
  AMHDST    hDistList,
  AMHPOL    hPolicy,
  AMHRCV    hReceiver
  AMLONG    dataLen,
  PAMBYTE   pData,
  AMHMSG    hMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hDistList	The distribution list handle returned by **amSesCreateDistList** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
hReceiver	The handle of the receiver service to which the response to this message should be sent, if the message being sent is a request message (input). Specify as AMH_NULL_HANDLE if no response is required.
dataLen	The length of the message data, in bytes (input). If set to zero, the data should be passed in the message object (`hMsg`).
pData	The message data (input).
hMsg	The handle of a message object that specifies the properties for the message being sent (input). If `dataLen` is zero, it should also contain the message data. If specified as AMH_NULL_HANDLE, the default send message object (constant: AMSD_SND_MSG_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amDstSendFile

Sends data from a file to each sender in the distribution list. The file data can be received as normal message data by a target application using **amRcvReceive** or used to reconstruct the file with **amRcvReceiveFile**.

```
AMBOOL amDstSendFile(
  AMHDST    hDistList,
  AMHPOL    hPolicy,
  AMLONG    options,
  AMLONG    directoryLen,
  AMSTR     directory,
  AMLONG    fileNameLen,
  AMSTR     fileName,
  AMHMSG    hMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hDistList	The distribution list handle returned by **amSesCreateDistList** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
options	Reserved, must be specified as 0L (input).
directoryLen	A reserved field that must be specified as zero (input).
directory	A reserved field that must be specified as NULL (input).
fileNameLen	The length of the file name in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
fileName	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the filename will travel with the message for use with a receive file call (see "amRcvReceiveFile" on page 131 for more details). Note that the filename sent will exactly match the supplied filename; it will not be converted or expanded in any way.
hMsg	The handle of the message object to use to send the file (input). This can be used to specify the Correlation ID for example. If specified as ANM_NULL_HANDLE, the default send message object (constant: AMSD_SND_MSG_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

If, in your application, you have previously used a message object, referenced by either handle or name, to send or receive data (including AMI elements or topics), you will need to explicitly call amMsgReset before re-using the object for sending a file. This applies even if you use the system default object handle (constant: AMSD_SND_MSG_HANDLE).

The system default message object handle is used when you specify `hMsg` as AMH_NULL_HANDLE.

Publisher interface functions

A *publisher* object encapsulates a sender object. It provides support for publish messages to a publish/subscribe broker.

amPubClearErrorCodes

Clears the error codes in the publisher object.

```
AMBOOL amPubClearErrorCodes(
  AMHPUB    hPublisher,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPublisher	The publisher handle returned by **amSesCreatePublisher** (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amPubClose

Closes the publisher service.

```
AMBOOL amPubClose(
  AMHPUB    hPublisher,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPublisher	The publisher handle returned by **amSesCreatePublisher** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amPubGetCCSID

Gets the coded character set identifier of the publisher service. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the publisher must perform CCSID conversion of the message before it is sent.

```
AMBOOL amPubGetCCSID(
  AMHPUB    hPublisher,
  PAMLONG   pCCSID,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPublisher	The publisher handle returned by **amSesCreatePublisher** (input).
pCCSID	The coded character set identifier (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amPubGetEncoding

Gets the value used to encode numeric data types for the publisher service. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the publisher must convert the encoding of the message before it is sent.

```
AMBOOL amPubGetEncoding(
  AMHPUB    hPublisher,
  PAMLONG   pEncoding,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPublisher	The publisher handle returned by **amSesCreatePublisher** (input).
pEncoding	The encoding (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amPubGetLastError

Gets the information (completion and reason codes) from the last error for the publisher object.

```
AMBOOL amPubGetLastError(
  AMHPUB    hPublisher,
  AMLONG    buffLen,
  PAMLONG   pStringLen,
  AMSTR     pErrorText,
  PAMLONG   pReason2,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPublisher	The publisher handle returned by **amSesCreatePublisher** (input).
buffLen	Reserved, must be zero (input).
pStringLen	Reserved, must be NULL (input).
pErrorText	Reserved, must be NULL (input).
pReason2	A secondary reason code (output). Not returned if specified as NULL. If `pReason` indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, `pReason2` gives a WebSphere MQ reason code.
pCompCode	Completion code (output). Not returned if specified as NULL.
pReason	Reason code (output). Not returned if specified as NULL. A value of AMRC_SERVICE_HANDLE_ERR indicates that the **amPubGetLastError** function call has itself detected an error and failed.

amPubGetName

Gets the name of the publisher service.

```
AMBOOL amPubGetName(
  AMHPUB    hPublisher,
  AMLONG    buffLen,
  PAMLONG   pNameLen,
  AMSTR     pName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPublisher	The publisher handle returned by **amSesCreatePublisher** (input).
buffLen	The length in bytes of a buffer into which the name is put (input). Set it to zero to return only the name length.
pNameLen	The length of the name, in bytes (output). Set it to NULL to return only the name.
pName	The publisher object name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amPubOpen

Opens the publisher service.

```
AMBOOL amPubOpen(
  AMHPUB    hPublisher,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPublisher	The publisher handle returned by **amSesCreatePublisher** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amPubPublish

Publishes a message using the publisher service.

The message data is passed in the message object. There is no option to pass it as a separate parameter as with **amSndSend** (this would not give any performance improvement because the MQRFH header has to be added to the message data before publishing it).

```
AMBOOL amPubPublish(
  AMHPUB    hPublisher,
  AMHPOL    hPolicy,
  AMHRCV    hReceiver,
  AMHMSG    hPubMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPublisher	The publisher handle returned by **amSesCreatePublisher** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
hReceiver	The handle of the receiver service to which the response to this publish request should be sent (input). Specify as AMH_NULL_HANDLE if no response is required. This parameter is mandatory if the policy specifies implicit registration of the publisher.
hPubMsg	The handle of a message object for the publication message (input). If specified as AMH_NULL_HANDLE, the default message object (constant: AMSD_SND_MSG_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Subscriber interface functions

A *subscriber* object encapsulates both a sender object and a receiver object. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

amSubClearErrorCodes

Clears the error codes in the subscriber object.

```
AMBOOL amSubClearErrorCodes(
  AMHSUB    hSubscriber,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubClose

Closes the subscriber service.

```
AMBOOL amSubClose(
  AMHSUB    hSubscriber,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubGetCCSID

Gets the coded character set identifier of the subscriber's sender service. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the subscriber must perform CCSID conversion of the message before it is sent.

```
AMBOOL amSubGetCCSID(
  AMHSUB    hSubscriber,
  PAMLONG   pCCSID,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
pCCSID	The coded character set identifier (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubGetDefnType

Gets the definition type of the subscriber's receiver service.

```
AMBOOL amSubGetDefnType(
  AMHSUB    hSubscriber,
  PAMLONG   pType,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).

pType	The definition type (output). It can be:
	`AMDT_UNDEFINED`
	`AMDT_TEMP_DYNAMIC`
	`AMDT_DYNAMIC`
	`AMDT_PREDEFINED`

pCompCode	Completion code (output).
pReason	Reason code (output).

amSubGetEncoding

Gets the value used to encode numeric data types for the subscriber's sender service. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the subscriber must convert the encoding of the message before it is sent.

```
AMBOOL amSubGetEncoding(
  AMHSUB    hSubscriber,
  PAMLONG   pEncoding,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
pEncoding	The encoding (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubGetLastError

Gets the information (completion and reason codes) from the last error for the subscriber object.

```
AMBOOL amSubGetLastError(
  AMHSUB    hSubscriber,
  AMLONG    buffLen,
  PAMLONG   pStringLen,
  AMSTR     pErrorText,
  PAMLONG   pReason2,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
buffLen	Reserved, must be zero (input).
pStringLen	Reserved, must be NULL (input).
pErrorText	Reserved, must be NULL (input).
pReason2	A secondary reason code (output). Not returned if specified as NULL. If `pReason` indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, `pReason2` gives a WebSphere MQ reason code.
pCompCode	Completion code (output). Not returned if specified as NULL.
pReason	Reason code (output). Not returned if specified as NULL. A value of AMRC_SERVICE_HANDLE_ERR indicates that the **amSubGetLastError** function call has itself detected an error and failed.

amSubGetName

Gets the name of the subscriber object.

```
AMBOOL amSubGetName(
  AMHSUB    hSubscriber,
  AMLONG    buffLen,
  PAMLONG   pNameLen,
  AMSTR     pName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
buffLen	The length in bytes of a buffer into which the name is put (input). Set it to zero to return only the name length.
pNameLen	The length of the name, in bytes (output). Set it to NULL to return only the name.
pName	The subscriber object name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubGetQueueName

Gets the queue name of the subscriber’s receiver service object. This can be used to determine the queue name of a permanent dynamic receiver service, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **amSubSetQueueName**.)

```
AMBOOL amSubGetQueueName(
  AMHSUB    hSubscriber,
  AMLONG    buffLen,
  PAMLONG   pStringLen,
  AMSTR     pQueueName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
buffLen	The length in bytes of a buffer in which the queue name is returned (input). Specify as zero to return only the length.
pStringLen	The length of the queue name, in bytes (output). If specified as NULL, the length is not returned.
pQueueName	The queue name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubOpen

Opens the subscriber service.

```
AMBOOL amSubOpen(
  AMHSUB    hSubscriber,
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubReceive

Receives a message, normally a publication, using the subscriber service. The message data, topic and other elements can be accessed using the message interface functions (see page 96).

The message data is passed in the message object. There is no option to pass it as a separate parameter as with **amRcvReceive** (this would not give any performance improvement because the MQRFH header has to be removed from the message data after receiving it).

```
AMBOOL amSubReceive(
  AMHSUB    hSubscriber,
  AMHPOL    hPolicy,
  AMHMSG    hSelMsg,
  AMHMSG    hRcvMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
hSelMsg	The handle of a selection message object (input). This is used to identify the message to be received (for example, using the correlation ID). Specify as AMH_NULL_HANDLE to get the next available message with no selection.
hRcvMsg	The handle of the message object for the received message (output). If specified as AMH_NULL_HANDLE, the default message object (constant: AMSD_RCV_MSG_HANDLE) is used. The message object is reset implicitly before the receive takes place.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubSetQueueName

Sets the queue name of the subscriber's receiver object, when this encapsulates a model queue. This can be used to specify the queue name of a recreated permanent dynamic receiver service, in order to receive messages in a session subsequent to the one in which it was created. (See also **amSubGetQueueName**.)

```
AMBOOL amSubSetQueueName(
  AMHSUB    hSubscriber,
  AMLONG    nameLen,
  AMSTR     pQueueName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
nameLen	The length of the queue name, in bytes (input).
pQueueName	The queue name (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubSubscribe

Sends a subscribe message to a publish/subscribe broker using the subscriber service, to register a subscription. The topic and other elements can be specified using the message interface functions (see page 96) before sending the message.

Publications matching the subscription are sent to the receiver service associated with the subscriber. By default, this has the same name as the subscriber service, with the addition of the suffix '.RECEIVER'.

```
AMBOOL amSubSubscribe(
  AMHSUB    hSubscriber,
  AMHPOL    hPolicy,
  AMHRCV    hReceiver,
  AMHMSG    hSubMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
hReceiver	The handle of the receiver service to which the response to this subscribe request should be sent (input). Specify as AMH_NULL_HANDLE if no response is required. This is not the service to which publications will be sent by the broker; they are sent to the receiver service associated with the subscriber (see above).
hSubMsg	The handle of a message object for the subscribe message (input). If specified as AMH_NULL_HANDLE, the default message object (constant: AMSD_SND_MSG_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubUnsubscribe

Sends an unsubscribe message to a publish/subscribe broker using the subscriber service, to deregister a subscription. The topic and other elements can be specified using the message interface functions (see page 96) before sending the message.

To deregister all topics, a policy providing this option must be specified (this is not the default policy). Otherwise, to remove a previous subscription the topic information specified must match that specified on the relevant **amSubSubscribe** request.

```
AMBOOL amSubUnsubscribe(
  AMHSUB    hSubscriber,
  AMHPOL    hPolicy,
  AMHRCV    hReceiver,
  AMHMSG    hUnsubMsg,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by **amSesCreateSubscriber** (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.
hReceiver	The handle of the receiver service to which the response to this unsubscribe request should be sent (input). Specify as AMH_NULL_HANDLE if no response is required.
hUnsubMsg	The handle of a message object for the unsubscribe message (input). If specified as AMH_NULL_HANDLE, the default message object (constant: AMSD_SND_MSG_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Policy interface functions

A *policy* object encapsulates the set of options used for each AMI request (open, close, send, receive, publish and so on). Examples are the priority and persistence of the message, and whether the message is included in a unit of work.

amPolClearErrorCodes

Clears the error codes in the policy object.

```
AMBOOL amPolClearErrorCodes(
  AMHPOL    hPolicy,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

`hPolicy`	The policy handle returned by **amSesCreatePolicy** (input).
`pCompCode`	Completion code (output).
`pReason`	Reason code (output).

amPolGetLastError

Gets the information (completion and reason codes) from the last error for the policy object.

```
AMBOOL amPolGetLastError(
  AMHPOL    hPolicy,
  AMLONG    buffLen,
  PAMLONG   pStringLen,
  AMSTR     pErrorText,
  PAMLONG   pReason2,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

`hPolicy`	The policy handle returned by **amSesCreatePolicy** (input).
`buffLen`	Reserved, must be zero (input).
`pStringLen`	Reserved, must be NULL (input).
`pErrorText`	Reserved, must be NULL (input).
`pReason2`	A secondary reason code (output). Not returned if specified as NULL. If `pReason` indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, `pReason2` gives a WebSphere MQ reason code.
`pCompCode`	Completion code (output). Not returned if specified as NULL.
`pReason`	Reason code (output). Not returned if specified as NULL. A value of AMRC_POLICY_HANDLE_ERR indicates that the **amPolGetLastError** function call has itself detected an error and failed.

C policy interface

amPolGetName

Returns the name of the policy object.

```
AMBOOL amPolGetName(
  AMHPOL    hPolicy,
  AMLONG    buffLen,
  PAMLONG   pNameLen,
  AMSTR     pName,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPolicy	The policy handle returned by **amSesCreatePolicy** (input).
buffLen	The length in bytes of a buffer into which the name is put (input). Set it to zero to return only the name length.
pNameLen	The length of the name, in bytes (output). Set it to NULL to return only the name.
pName	The policy object name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amPolGetWaitTime

Returns the wait time (in ms) set for this policy.

```
AMBOOL amPolGetWaitTime(
  AMHPOL    hPolicy,
  PAMLONG   pWaitTime,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPolicy	The policy handle returned by **amSesCreatePolicy** (input).
pWaitTime	The wait time, in ms (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amPolSetWaitTime

Sets the wait time for any receive function using this policy.

```
AMBOOL amPolSetWaitTime(
  AMHPOL    hPolicy,
  AMLONG    waitTime,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPolicy	The policy handle returned by **amSesCreatePolicy** (input).
waitTime	The wait time (in ms) to be set in the policy (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

Chapter 6. C policy handler interface

Policy handler libraries are collections of functions, implemented in C, that are:
- external to the AMI
- called from predefined AMI invocation points

The policy handler framework allows the AMI to be extended to include additional message handling functions beyond those that are in the product. Examples of such additional functions include:
- writing audit records to a database or audit queue
- adding additional information to the message, for example, inserting headers
- compressing message data
- enforcing corporate messaging standards
- interfacing to a system management tool to report error conditions
- sending or receiving messages using alternative transport protocols

Vendors might provide policy handlers that are generally applicable, or customers might create policy handlers for their own specific requirements.

A policy handler is invoked when a policy used on an AMI function call is configured to specify the policy handler. More than one policy handler can be invoked on a single AMI call, and they are invoked in the order specified in the policy repository definition.

A policy handler can be called at various points during AMI processing. When the policy handler is initialized, it specifies to the AMI the list of invocation points at which it can be called.

Static initialization and invocation parameter strings can be specified for the policy handler in the repository. An additional parameter string, for use by the policy handler, can be specified for a service point.

Existing application programs that already use policy definitions in the repository (rather than a system default policy) can exploit policy handler library functions. To do this, you simply update the AMI repository by using the AMI Administration Tool. You do not need to change the application program.

To define a policy handler, you use the AMI Administration Tool to associate an administrator-defined name with the file name of the policy handler library. In an existing policy object definition, you can specify the list of policy handlers to associate with this policy. To do this, in the AMI Administration Tool, you access the policy and its Handlers tab, then select from the list of existing policy handlers (defined using the policy handler definitions pane).

A policy handler library can create and manage its own state information in a library context object. To do this, it returns a context handle on creation that is passed to it on each subsequent call. During initialization, the library must register each invocation point it wishes to support in its implementation.

C policy handler interface

Note the following features of policy handlers:

- There can be more than one policy handler library definition in the repository.
- It is possible to specify a sequence of different policy handler invocations in a policy for a single invocation point.
- It is possible to specify three separate static parameter strings in the repository that can be passed to the policy handler to further customize its operation.

  One parameter string is passed on initialization, and the other two are passed at each invocation point. These can provide information that is specific to the current policy, and information that is specific to the current sender or receiver service object.

The next sections describe invocation points and the functions required for policy handler operation. For further information about policy handlers, see:

- "Writing policy handlers in C" on page 38
- "The AMI policy handler sample program (amtsphlr)" on page 487
- "Policy definitions" on page 497 and "Policy handler definitions" on page 504
- Appendix C, "Extended C AMI functions" on page 591

Invocation points

Policy handler invocation points are predefined AMI exits, from where you can optionally invoke a policy handler function. Invocation points are provided at two levels:

- AMI operation invocation
- Transport invocation

For each AMI operation, there is a single AMI operation invocation point plus one or more transport invocation points. For example, if you open a session, this opens a connection that has an open connection (AMI) invocation point, and MQCONNX, MQOPEN, and MQINQ (transport) invocation points.

AMI operation invocations

AMI operation invocations are calls that occur at the start of each AMI processing operation that involves any transport request. Typically, these might be used to replace the existing (WebSphere MQ) transport implementation with an alternative transport implementation. Therefore, they allow you to provide alternative pluggable transport drivers alongside WebSphere MQ. The supported set of AMI operation invocation points are as follows:

```
Open Connection          (AMINV_CONNECTION_OPEN)
Close Connection         (AMINV_CONNECTION_CLOSE)
Begin                    (AMINV_BEGIN)
Rollback                 (AMINV_ROLLBACK)
Commit                   (AMINV_COMMIT)
Open Sender              (AMINV_SENDER_OPEN)
Close Sender             (AMINV_SENDER_CLOSE)
Send To Sender           (AMINV_SENDER_SEND)
Open Receiver            (AMINV_RECEIVER_OPEN)
Close Receiver           (AMINV_RECEIVER_CLOSE)
Receiver From Receiver   (AMINV_RECEIVER_RECEIVE)
Open Distribution List   (AMINV_DIST_LIST_OPEN)
Close Distribution List  (AMINV_DIST_LIST_CLOSE)
Send To Distribution List (AMINV_DIST_LIST_SEND)
Handle Poison Message    (AMINV_HANDLE_POISON_MSG)
```

Pre-transport request invocations

Pre-transport request invocations are calls that occur immediately before each underlying WebSphere MQ transport call. These calls are as follows:

```
Pre-MQBACK  (AMINV_PRE_MQBACK)
Pre-MQBEGIN (AMINV_PRE_MQBEGIN)
Pre-MQCLOSE (AMINV_PRE_MQCLOSE)
Pre-MQCMIT  (AMINV_PRE_MQCMIT)
Pre-MQCONN  (AMINV_PRE_MQCONN)
Pre-MQCONNX (AMINV_PRE_MQCONNX)
Pre-MQDISC  (AMINV_PRE_MQDISC)
Pre-MQGET   (AMINV_PRE_MQGET)
Pre-MQINQ   (AMINV_PRE_MQINQ)
Pre-MQOPEN  (AMINV_PRE_MQOPEN)
Pre-MQPUT   (AMINV_PRE_MQPUT)
Pre-MQPUT1  (AMINV_PRE_MQPUT1)
Pre-MQSET   (AMINV_PRE_MQSET)
```

Post-transport request invocations

Post-transport request invocations are calls that occur immediately after each underlying WebSphere MQ transport call. These calls are as follows:

```
Post-MQBACK  (AMINV_POST_MQBACK)
Post-MQBEGIN (AMINV_POST_MQBEGIN)
Post-MQCLOSE (AMINV_POST_MQCLOSE)
Post-MQCMIT  (AMINV_POST_MQCMIT)
Post-MQCONN  (AMINV_POST_MQCONN)
Post-MQCONNX (AMINV_POST_MQCONNX)
Post-MQDISC  (AMINV_POST_MQDISC)
Post-MQGET   (AMINV_POST_MQGET)
Post-MQINQ   (AMINV_POST_MQINQ)
Post-MQOPEN  (AMINV_POST_MQOPEN)
Post-MQPUT   (AMINV_POST_MQPUT)
Post-MQPUT1  (AMINV_POST_MQPUT1)
Post-MQSET   (AMINV_POST_MQSET)
```

Policy handler library functions

A policy handler library must implement the following functions:

- “amPhlCreate” on page 154
- “amPhlDelete” on page 154
- “amPhlInitialize” on page 154
- “Invocation point functions (amPhl*Xxx*)” on page 155

Note that all functions return a completion code (`pCompCode`) and a policy handler reason code (`pReason`). The completion code can take one of the following values:

AMCC_OK	Function completed successfully
AMCC_WARNING	Function completed with a warning
AMCC_FAILED	An error occurred during processing

The implementor of the policy handler defines the reason code values that are returned by a specific policy handler library. The value returned must be AMRC_NONE with a completion code of AMCC_OK, or non-zero with a completion code of AMCC_WARNING or AMCC_ERROR.

Policy handler library functions

amPhlCreate

Creates a library context (data area) instance and returns its handle. This function is called at session creation time. There is one library context object for a session for each policy handler library definition in the repository.

amPhlCreate returns the library context handle (with a completion code of AMCC_OK or AMCC_WARNING). If there is an error, **amPhlCreate** returns AMH_NULL_HANDLE with a completion code of AMCC_FAILED.

```
AMHPHL AMENTRY amPhlCreate(
  AMLONG      type,
  AMLONG      nameLen,
  AMSTR       name,
  PAMLONG     pCompCode,
  PAMLONG     pReason);
```

type	Reserved, must be zero.
nameLen	The length of the name in bytes, excluding any terminating null (input). A value of AMLEN_NULL_TERM specifies that the name is a null-terminated string.
name	The policy handler name that the administrator assigned to this library in the repository (input).
pCompCode	Completion code (output).
pReason	Policy handler reason code (output).

amPhlDelete

Deletes the specified instance of the library context.

amPhlDelete returns AMB_TRUE for success (that is, with a completion code of AMCC_OK or AMCC_WARNING), or AMB_FALSE if there is an error (that is, with a completion code of AMCC_FAILED).

```
AMBOOL AMENTRY amPhlDelete(
  PAMHPHL     phContext,
  PAMLONG     pCompCode,
  PAMLONG     pReason);
```

phContext	A pointer to the library context handle (input). This should be set to AMH_INVALID_HANDLE after the policy handler has been deleted and before returning from this function.
pCompCode	Completion code (output).
pReason	Policy handler reason code (output).

amPhlInitialize

Initializes the policy handler library. Before returning, this function should call back into the AMI using **amLibRegisterFunction** to register each of the invocation points that the policy handler wishes to support in its implementation.

amPhlInitialize returns AMB_TRUE for success (that is, with a completion code of AMCC_OK or AMCC_WARNING), or AMB_FALSE if there is an error (that is, with a completion code of AMCC_FAILED).

```
AMBOOL AMENTRY amPhlInitialize(
  AMHPHL      hContext,
  AMHLIB      hLibrary,
  AMLONG      initParmLen,
  AMSTR       initParms,
  AMBOOL      traceOn,
  PAMLONG     pCompCode,
  PAMLONG     pReason);
```

hContext	The library context handle (input).
hLibrary	The AMI library handle to be used with amLibRegisterFunction (input).
initParmLen	The length of the initialization parameter string that is defined in the repository for this policy handler (excluding any terminating null).
initParms	The initialization parameter string that is defined in the repository definition for this policy handler (input). The string is NULL-terminated.
traceOn	Set tracing on (input). This must be specified as AMB_TRUE or AMB_FALSE. If tracing is on, the policy handler should call back into the AMI by using **amLibTraceText** to add trace entries to the AMI trace buffer. Trace entry and exit points should be included at the start and end of each function. They should also be included at any other critical point in the code where additional information may help in problem diagnosis. Examples include before and after calls to other components. Trace strings should include any information that may be useful for problem diagnosis, including information about error codes and return codes.
pCompCode	Completion code (output).
pReason	Policy handler reason code (output).

Invocation point functions (amPhl*Xxx*)

A policy handler library must implement one or more invocation point functions, with the following function prototype, for those functions that the policy handler library implements.

A policy handler can use the same function for multiple invocation points, or use a separate function for each.

The implementor of the policy handler library determines the function names. These function names are defined by, and unique to, an individual policy handler library.

amPhl*Xxx* returns AMB_TRUE for success (that is, with completion code AMCC_OK or AMCC_WARNING), or AMB_FALSE if there is an error (that is, with completion code AMCC_FAILED).

Policy handler library functions

```
AMBOOL AMENTRY amPhlXxx(
  AMHPHL      hContext,
  AMLONG      transportLen,
  AMSTR       transport,
  AMLONG      invocationPoint,
  AMLONG      customParmLen,
  AMSTR       customParms,
  AMLONG      invParmLen,
  AMSTR       invParms,
  AMLONG      amiOperation,
  PAMPHPARM   pAmiParms,
  AMPTR       pTrpParms,
  PAMLONG     pContinue,
  PAMLONG     pCompCode,
  PAMLONG     pReason);
```

hContext The library context handle (input).

transportLen The length of the transport name in bytes. This must be 9 (see AMPH_TRANSPORT_LEN_MQ in the C header file amtphlc.h).

transport The transport name. This must be "WebSphere MQ" (see AMPH_TRANSPORT_TYPE_MQ in the C header file amtphlc.h).

invocationPoint
The invocation point, specified as one of the AMINV_ values listed in "Invocation points" on page 152 (input).

customParmLen The length of the custom parameter string that is defined in the repository for this service point, excluding any terminating null (input).

customParms The custom parameter string that is defined in the repository for this service point (input). The string is NULL-terminated.

invParmLen The length of the invocation parameter string that is defined in the repository for this policy, excluding any terminating null (input).

invParms The invocation parameter string that is defined in the repository for this policy (input). The string is NULL-terminated.

amiOperation The current AMI operation from which this invocation originates (input). This can be one of:

```
AMOP_CONNECTION_OPEN:     open transport connection
AMOP_CONNECTION_CLOSE:    close transport connection
AMOP_BEGIN:               begin unit of work
AMOP_ROLLBACK:            roll back unit of work
AMOP_COMMIT:              commit unit of work
AMOP_SENDER_OPEN:         open sender
AMOP_SENDER_CLOSE:        close sender
AMOP_SENDER_SEND:         send to sender
AMOP_RECEIVER_OPEN:       open receiver
AMOP_RECEIVER_CLOSE:      close receiver
AMOP_RECEIVER_RECEIVE:    receive from receiver
AMOP_DIST_LIST_OPEN:      open distribution list
AMOP_DIST_LIST_CLOSE:     close distribution list
AMOP_DIST_LIST_SEND:      send to distribution list
AMOP_HANDLE_POISON_MSG:   handle poison message
```

amiOperation defines the information specified by the **pAmiParms** parameters below, and can be used to tailor the invocation point processing to the particular AMI operation. For example, AMINV_PRE_MQOPEN with AM_SENDER_OPEN might require different processing from AMINV_PRE_MQOPEN with AMOP_RECEIVER_OPEN.

pAmiParms The parameters of the current AMI operation (input). This is a pointer to a union, where the contents of that union represent a structure that is determined by the **amiOperation** parameter, as follows:

AMI Operation	Structure
AMOP_CONNECTION_OPEN:	AMPHOPC
AMOP_CONNECTION_CLOSE:	AMPHCLC
AMOP_BEGIN:	AMPHBGN
AMOP_ROLLBACK:	AMPHRBK
AMOP_COMMIT:	AMPHCMT
AMOP_SENDER_OPEN:	AMPHOPS
AMOP_SENDER_CLOSE:	AMPHCLS
AMOP_SENDER_SEND:	AMPHSNS
AMOP_RECEIVER_OPEN:	AMPHOPS
AMOP_RECEIVER_CLOSE:	AMPHCLS
AMOP_RECEIVER_RECEIVE:	AMPHRCS
AMOP_DIST_LIST_OPEN:	AMPHOPD
AMOP_DIST_LIST_CLOSE:	AMPHCLD
AMOP_DIST_LIST_SEND:	AMPHSND
AMOP_HANDLE_POISON_MSG:	AMPHHPM

pTrpParms The parameters of the current transport operation for the invocation point (input). This is a pointer to a union, where the contents of that union represent a structure that is determined by the **invocationPoint** parameter as follows:

Invocation point	Structure
AMINV_CONNECTION_OPEN:	None (NULL pointer)
AMINV_CONNECTION_CLOSE:	None (NULL pointer)
AMINV_BEGIN:	None (NULL pointer)
AMINV_ROLLBACK:	None (NULL pointer)
AMINV_COMMIT:	None (NULL pointer)
AMINV_SENDER_OPEN:	None (NULL pointer)
AMINV_SENDER_CLOSE:	None (NULL pointer)
AMINV_SENDER_SEND:	None (NULL pointer)
AMINV_RECEIVER_OPEN:	None (NULL pointer)
AMINV_RECEIVER_CLOSE:	None (NULL pointer)
AMINV_RECEIVER_RECEIVE:	None (NULL pointer)
AMINV_DIST_LIST_OPEN:	None (NULL pointer)
AMINV_DIST_LIST_CLOSE:	None (NULL pointer)
AMINV_DIST_LIST_SEND:	None (NULL pointer)
AMINV_HANDLE_POISON_MSG:	None (NULL pointer)
AMINV_PRE_MQBACK:	AMPHMQBACK
AMINV_POST_MQBACK:	AMPHMQBACK
AMINV_PRE_MQBEGIN:	AMPHMQBEGIN
AMINV_POST_MQBEGIN:	AMPHMQBEGIN
AMINV_PRE_MQCLOSE:	AMPHMQCLOSE
AMINV_POST_MQCLOSE:	AMPHMQCLOSE
AMINV_PRE_MQCMIT:	AMPHMQCMIT
AMINV_POST_MQCMIT:	AMPHMQCMIT
AMINV_PRE_MQCONN:	AMPHMQCONN
AMINV_POST_MQCONN:	AMPHMQCONN
AMINV_PRE_MQCONNX:	AMPHMQCONNX
AMINV_POST_MQCONNX:	AMPHMQCONNX
AMINV_PRE_MQDISC:	AMPHMQDISC
AMINV_POST_MQDISC:	AMPHMQDISC
AMINV_PRE_MQGET:	AMPHMQGET
AMINV_POST_MQGET:	AMPHMQGET
AMINV_PRE_MQINQ:	AMPHMQINQ
AMINV_POST_MQINQ:	AMPHMQINQ
AMINV_PRE_MQOPEN:	AMPHMQOPEN
AMINV_POST_MQOPEN:	AMPHMQOPEN
AMINV_PRE_MQPUT:	AMPHMQPUT
AMINV_POST_MQPUT:	AMPHMQPUT

```
AMINV_PRE_MQPUT1:         AMPHMQPUT1
AMINV_POST_MQPUT1:        AMPHMQPUT1
AMINV_PRE_MQSET:          AMPHMQSET
AMINV_POST_MQSET:         AMPHMQSET
```

pContinue Returns the continuation code as one of:

AMPH_CONTINUE:
Continue processing. The AMI continues with normal processing on return from the policy handler function call. If the invocation returns AMB_FALSE, indicating that the request failed, subsequent AMI processing is limited to normal error handling. With this continuation code, any output information (except for error information) that is returned via the call parameters is ignored.

AMPH_COMPLETE:
Invocation is complete. The behavior differs, depending on the context of the invocation:

- Context: AMI Operation Invocation

  This continuation code indicates that the policy handler has completed all processing for the current AMI operation. Subsequent processing is limited to error handling only.
- Context: Pre-Transport Invocation

  This continuation code indicates that the policy handler has completed all processing for the WebSphere MQ transport request for the current invocation point. The AMI skips the transport request and continues with post-transport invocation, where applicable. If the policy handler returns AMB_FALSE, indicating that the request failed, subsequent AMI processing is limited to normal error handling.
- Context: Post-transport AMI function invocation

  The continuation code is ignored. The AMI continues with normal processing. If the policy handler returns AMB_FALSE, indicating that the request failed, subsequent AMI processing is limited to normal error handling.

pCompCode Completion code (output).

pReason Policy handler reason code (output).

AMI extensions for policy handler callback functions

The following functions represent the AMI extensions that are provided to enable the policy handler to make the necessary AMI callback requests.

Note that all functions return a completion code (`pCompCode`) and an AMI reason code (`pReason`). The completion code can take one of the following values:

AMCC_OK	Function completed successfully
AMCC_WARNING	Function completed with a warning
AMCC_FAILED	An error occurred during processing

If the completion code returns warning or failed, the reason code identifies the reason for the error or warning (see Appendix A, “Reason codes and LDAP error codes” on page 537).

amLibRegisterFunction

This function is called during **amPhlInitialize** processing to register an invocation point function.

```
AMBOOL AMENTRY amLibRegisterFunction(
  AMHLIB      hLibrary,
  AMLONG      invocationPoint,
  PAMFUNC     pFunction,
  PAMLONG     pCompCode,
  PAMLONG     pReason);
```

amLibRegisterFunction returns AMB_TRUE for success, or AMB_FALSE if there is an error.

`hLibrary`	The AMI library handle parameter used with **amPhlInitialize**.
`invocationPoint`	The invocation point for this function pointer, specified as one of the AMINV_ values listed in “Invocation points” on page 152 (input).
`pFunction`	The function address for this invocation point.
`pCompCode`	Completion code (output).
`pReason`	AMI Reason code (output).

amLibTraceText

This function traces the specified text string and adds it to the AMI trace buffer if tracing is on. If tracing is off, it does nothing.

```
AMBOOL AMENTRY amLibTraceText(
  AMLONG      control,
  AMLONG      stringLength,
  AMSTR       string,
  PAMLONG     pCompCode,
  PAMLONG     pReason);
```

`control` Control information, which can be one of the following values:

`AMTC_TEXT:`
Trace text string.

`AMTC_FUNCTION_ENTRY:`
Trace function entry (the function name should be included in the text string parameter).

AMTC_FUNCTION_EXIT:
Trace function exit (the function name and any return code value should be included in the text string parameter).

AMTC_DEFAULT:
The same as AMTC_TEXT, that is, trace text string.

stringLength	The length of the text string name in bytes (excluding any terminating null).
string	The text string to be traced.
pCompCode	Completion code (output).
pReason	AMI Reason code (output).

Part 3. The C++ interface

Chapter 7. Using the Application Messaging Interface in C++

The Application Messaging Interface for C++ (amCpp) provides a C++ style of programming, while being consistent with the object-style interface of the Application Messaging Interface for C.

This chapter describes the following:
- "Structure of the AMI"
- "Writing applications in C++" on page 167
- "Building C++ applications" on page 177

Note that the term *object* is used in this book in the object-oriented programming sense, not in the sense of WebSphere MQ 'objects' such as channels and queues.

Structure of the AMI

The following classes are provided:

Base classes

AmSessionFactory	Creates AmSession objects.
AmSession	Creates objects within the AMI session, and controls transactional support.
AmMessage	Contains the message data, message ID and correlation ID, and options that are used when sending or receiving a message (most of which come from the policy definition).
AmSender	This is a service that represents a destination (such as a WebSphere MQ queue) to which messages are sent.
AmReceiver	This is a service that represents a source (such as a WebSphere MQ queue) from which messages are received.
AmDistributionList	Contains a list of sender services to provide a list of destinations.
AmPublisher	Contains a sender service where the destination is a publish/subscribe broker.
AmSubscriber	Contains a sender service (to send subscribe and unsubscribe messages to a publish/subscribe broker) and a receiver service (to receive publications from the broker).
AmPolicy	Defines how the message should be handled, including items such as priority, persistence, and whether it is included in a unit of work.

Interface and helper classes

AmObject	This is an abstract class, from which the base classes listed previously inherit (with the exception of AmSessionFactory).
AmElement	This encapsulates name/value pairs for use in publish/subscribe applications.
AmStatus	This encapsulates the error status of amCpp objects.
AmString	This encapsulates string data.
AmBytes	This encapsulates binary/byte data.

Exception classes

AmException	This is the base Exception class for amCpp; all other amCpp Exceptions inherit from this class.
AmErrorException	An Exception of this type is raised when an amCpp object experiences an error with a severity level of FAILED (CompletionCode = AMCC_FAILED).
AmWarningException	An Exception of this type is raised when an amCpp object experiences an error with a severity level of WARNING (CompletionCode = AMCC_WARNING), provided that warnings have been enabled using the **enableWarnings** method.

Using the repository

You can run AMI applications with or without a repository. If you do not have a repository, you can create an object by specifying its name in a method. It will be created using the appropriate system provided definition (see “System provided definitions” on page 492).

If you have a repository, and you specify the name of an object in a method that matches a name in the repository, the object will be created using the repository definition. (If no matching name is found in the repository, the system provided definition will be used.)

System default objects

The set of system default objects created in C is not accessible directly in C++, but the SYSTEM.DEFAULT.POLICY (constant: AMSD_POL) is used to provide default behavior when a policy is not specified. Objects with identical properties to the system default objects can be created for use in C++ using the built-in definitions (see “System provided definitions” on page 492).

Writing applications in C++

This section gives a number of examples showing how to access the Application Messaging Interface using C++.

Many of the method calls are overloaded and in some cases this results in default objects being used. One example of this is the AmPolicy object which can be passed on many of the methods. For example:

Method overloading

```
mySender->send(*mySendMessage, *myPolicy);

mySender->send(*mySendMessage);
```

If a policy has been created to provide specific send behavior, use the first example. However, if the default policy is acceptable, use the second example.

The defaulting of behavior using method overloading is used throughout the examples.

Creating and opening objects

Before using the AMI, you must create and open the required objects. Objects are created with names, which might correspond to named objects in the repository. In the case of the creation of a response sender (`myResponder`) in the following example, the default name for a response type object is specified, so the object is created with default responder values.

Creating AMI objects

```
mySessionFactory = new AmSessionFactory("MY.REPOSITORY.XML");
mySession = mySessionFactory->createSession("MY.SESSION");
myPolicy = mySession->createPolicy("MY.POLICY");

mySender = mySession->createSender("AMT.SENDER.QUEUE");
myReceiver = mySession->createReceiver("AMT.RECEIVER.QUEUE");
myResponder = mySession->createSender(AMDEF_RSP_SND);

mySendMessage = mySession->createMessage("MY.SEND.MESSAGE");
myReceiveMessage = mySession->createMessage("MY.RECEIVE.MESSAGE");
```

The objects are then opened. In the following examples, the session object is opened with the default policy, whereas the sender and receiver objects are opened with a specified policy (`myPolicy`).

Opening the AMI objects

```
mySession->open();
mySender->open(*myPolicy);
myReceiver->open(*myPolicy);
```

Deleting objects

To avoid memory leaks, it is essential to explicitly delete all C++ objects that you have created at the end of your program. Delete the session after everything other than the session factory. Delete the session factory last.

The following is an example from the `Receiver.cpp` sample program:

Deleting AMI objects

```
mySession->deleteMessage(myReceiveMsg);
mySession->deleteReceiver(myReceiver);
mySession->deletePolicy(myPolicy);
mySessionFactory->deleteSession(mySession);
delete *mySessionFactory;
```

Sending messages

The examples in this section show how to send a datagram (send and forget) message. First, the message data is written to the `mySendMessage` object. Data is always sent in byte form using the AmBytes helper class.

Writing data to a message object

```
AmBytes *dataSent = new AmBytes((const char*)"message to be sent");
mySendMessage->writeBytes(*dataSent);
```

Next, the message is sent using the sender service `mySender`.

Sending a message

```
mySender->send(*mySendMessage);
```

The policy used is either the default policy for the service, if specified, or the system default policy. The message attributes are set from the policy or service, or the default for the messaging transport.

When more control is needed, you can pass a policy object:

Sending a message with a specified policy

```
mySender->send(*mySendMessage, *myPolicy);
```

The policy controls the behavior of the send command. In particular, the policy specifies whether the send is part of a unit of work, the priority, persistence and expiry of the message and whether policy components should be invoked. Whether the queue should be implicitly opened and left open can also be controlled.

To send a message to a distribution list, for example `myDistList`, use it as the sender service:

Sending a message to a distribution list

```
myDistList->send(*mySendMessage);
```

You can set an attribute such as the *`Format`* before a message is sent, to override the default in the policy or service.

Setting an attribute in a message

```
mySendMessage->setFormat("MyFormat"):
```

Similarly, after a message has been sent you can retrieve an attribute such as the *`MessageID`*. Binary data, such as *`MessageId`* can be extracted using the AmBytes helper class.

Getting an attribute from a message

```
AmBytes msgId = mySendMessage.getMessageId();
```

For details of the message attributes that you can set and get, see "AmMessage" on page 187.

When a message object is used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see "reset" on page 211) and rebuild it each time.

Sample program

For more details, refer to the `SendAndForget.cpp` sample program (see "Sample programs for iSeries, UNIX, and Windows" on page 480).

Receiving messages

The next example shows how to receive a message from the receiver service `myReceiver`, and to read the data from the message object `myReceiveMessage`.

Receiving a message and retrieving the data

```
myReceiver->receive(*myReceiveMessage);
AmBytes data = myReceiveMessage->readBytes(
                 myReceiveMessage->getDataLength());
```

The policy used will be the default for the service if defined, or the system default policy. Greater control of the behavior of the receive can be achieved by passing a policy object.

Receiving a message with a specified policy

```
myReceiver->receive(*myReceiveMessage, *myPolicy);
```

The policy can specify the wait interval, whether the call is part of a unit of work, whether the message should be code page converted, whether all the members of a group must be there before any members can be read, and how to deal with backout failures.

To receive a specific message using its correlation ID, create a selection message object and set its *CorrelId* attribute to the required value. The selection message is then passed as a parameter on the receive.

Receiving a specific message using the correlation ID

```
AmBytes * myCorrelId = new AmBytes("MYCORRELATION");
mySelectionMessage = mySession->createMessage("MY.SELECTION.MESSAGE");
mySelectionMessage->setCorrelationId(*myCorrelId);
myReceiver->receive(*myReceiveMessage, *mySelectionMessage, *myPolicy);
```

As before, the policy is optional.

You can view the attributes of the message just received, such as the *Encoding*.

Getting an attribute from the message

```
encoding = myReceiveMessage->getEncoding();
```

Sample program

For more details, refer to the `Receiver.cpp` sample program (see “Sample programs for iSeries, UNIX, and Windows” on page 480).

Request/response messaging

In the *request/response* style of messaging, a requester (or client) application sends a request message and expects to receive a response message back. The responder (or server) application receives the request message and produces the response message (or messages) which it sends back to the requester application. The responder application uses information in the request message to know how to send the response message back to the requester.

In the following examples, ‘my’ refers to the requesting application (the client) and ‘your’ refers to the responding application (the server).

The requester sends a message as described in “Sending messages” on page 168, specifying the service (`myReceiver`) to which the response message should be sent.

Sending a request message

```
mySender->send(*mySendMessage, *myReceiver);
```

A policy object can also be specified if required.

The responder receives the message as described in “Receiving messages” on page 169, using its receiver service (`yourReceiver`). It also receives details of the response service (`yourResponder`) for sending the response.

Receiving the request message

```
yourReceiver->receive(*yourReceiveMessage, *yourResponder);
```

A policy object can be specified if required, as can a selection message object (see “Receiving messages” on page 169).

The responder sends its response message (`yourReplyMessage`) to the response service, specifying the received message to which this is a response.

Sending a response to the request message

```
yourResponder->send(*yourReplyMessage, *yourReceiveMessage);
```

Finally, the requester application receives the response (`myResponseMessage`), which is correlated with the original message it sent (`mySendMessage`).

Receiving the response message

```
myReceiver->receive(*myResponseMessage, *mySendMessage);
```

In a typical application, the responder might be a server operating in a loop, receiving requests and replying to them. In this case, the message objects should be set to their initial state and the data cleared before servicing the next request. This is achieved as follows:

Resetting the message object

```
yourReceiveMessage->reset();
yourResponseMessage->reset();
```

Sample programs

For more details, refer to the `Client.cpp` and `Server.cpp` sample programs (see “Sample programs for iSeries, UNIX, and Windows” on page 480).

File transfer

You can perform file transfers using the `AmSender.sendFile` and `AmReceiver.receiveFile` methods.

Sending a file using the sendFile method

```
mySender->sendFile(*mySendMessage,myfilename,*myPolicy)
```

Receiving a file using the receiveFile method

```
myReceiver->receiveFile(*myReceiveMessage,myfileName,*myPolicy)
```

For a complete description of file transfer, refer to “File transfer” on page 21.

Publish/subscribe messaging

With *publish/subscribe* messaging, a *publisher* application publishes messages to *subscriber* applications using a *broker*. The message published contains application data and one or more *topic* strings that describe the data. A subscribing application subscribes to topics, informing the broker which topics it is interested in. When the broker receives a message from a publisher, it compares the topics in the messages to the topics in the subscription from subscribing applications. If they match, the broker forwards the message to the subscribing application.

Data on a particular topic is published as shown in the next example.

Publishing a message on a specified topic

```
AmBytes *publicationData = new AmBytes("The weather is sunny");

myPubMessage->addTopic("Weather");
myPubMessage->writeBytes(publicationData);
myPublisher->publish(*myPubMessage, *myReceiver);
```

`myReceiver` identifies a response service to which the broker will send any response messages (indicating whether the publish was successful or not). You can also specify a policy object to modify the behavior of the command.

To subscribe to a publish/subscribe broker you need to specify one or more topics.

Subscribing to a broker on specified topics

```
mySubMessage->addTopic("Weather");
mySubMessage->addTopic("Birds");
mySubscriber->subscribe(*mySubMessage, *myReceiver);
```

Broker response messages will be sent to `myReceiver`.

To remove a subscription, add the topic or topics to be deleted to the message object, and use:

Removing a subscription

```
mySubscriber->unsubscribe(*myUnsubMessage, *myReceiver);
```

To receive a publication from a broker, use:

Receiving a publication

```
mySubscriber->receive(*myReceiveMessage, *myPolicy);
publication = myReceiveMessage->readBytes(
                *myReceiveMessage->getDataLength());
```

You can then use the **getTopicCount** and **getTopic** methods to extract the topic or topics from the message object.

Subscribing applications can also exploit content-based publish/subscribe by passing a filter on subscribe and unsubscribe calls (see “Using WebSphere MQ Integrator Version 2” on page 477).

Sample programs

For more details, refer to the `Publisher.cpp` and `Subscriber.cpp` sample programs (see “Sample programs for iSeries, UNIX, and Windows” on page 480).

Using AmElement objects

Publish/subscribe brokers (such as WebSphere MQ Publish/Subscribe) respond to messages that contain name/value pairs to define the commands and options to be carried out. The Application Messaging Interface contains some methods which produce these name/value pairs directly (such as **AmSubscriber->subscribe**). For less commonly used commands, the name/value pairs can be added to a message using an AmElement object.

For example, to send a message containing a ‘Request Update’ command, use the following:

Using an AmElement object to construct a command message

```
AmElement *bespokeElement = new AmElement("MQPSCommand", "ReqUpdate");
mySendMessage->addElement(*bespokeElement);
```

You must then send the message, using **AmSender->send**, to the sender service specified for your publish/subscribe broker.

If you use streams with WebSphere MQ Publish/Subscribe, you must add the appropriate name/value element explicitly to the message object.

The message element methods can, in fact, be used to add any element to a message before issuing an publish/subscribe request. Such elements (including topics, which are specialized elements) supplement or override those added implicitly by the request, as appropriate to the individual element type.

The use of name/value elements is not restricted to publish/subscribe applications. They can be used in other applications as well.

Error handling

The **getLastErrorStatus** method always reflects the last most severe error experienced by an object. It can be used to return an AmStatus object encapsulating this error state. Once the error state has been handled, **clearErrorCodes** can be called to reset this error state.

AmCpp can raise two types of Exception, one to reflect serious errors and the other to reflect warnings. By default, only AmErrorExceptions are raised. AmWarningExceptions can be enabled using the **enableWarnings** method. Because both are types of AmException, a generic catch block can be used to process all amCpp Exceptions.

Enabling AmWarningExceptions might have some unexpected side-effects, especially when an AmObject is returning data such as another AmObject. For example, if AmWarningExceptions are enabled for an AmSession object and an AmSender is created that does not exist in the repository, an AmWarningException will be raised to reflect this fact. If this happens, the AmSender object will not be created, because its creation was interrupted by an Exception. However, there might be times during the life of an AmObject when processing AmWarningExceptions is useful.

For example:

```
try
{
    ...
    mySession->enableWarnings(AMB_TRUE);
    mySession->open();
    ...
}
catch (AmErrorException &errorEx)
{
    AmStatus sessionStatus = mySession->getLastErrorStatus();
    switch (sessionStatus.getReasonCode())
    {
    case AMRC_XXXX:
        ...
    case AMRC_XXXX:
        ...
    }
    mySession->clearErrorCodes();
}
catch (AmWarningException &warningEx)
{
    ...
}
```

Because most of the objects are types of AmObject, a generic error handling routine can be written. For example:

```
try
{
    ...
    mySession->open();
    ...
    mySender->send(*myMessage):
    ...
    mySender->send(*myMessage):
    ...
    mySession->commit();
}
catch(AmException &amex);
{
    AmStatus status = amex.getSource()->getLastErrorStatus();
    printf("Object in error; name = %s\n", amex.getSource()->getName());
    printf("Object in error; RC = %ld\n",  status.getReasonCode());
    ...
    amex.getSource()->clearErrorCodes();
}
```

The catch block works because all objects that throw the AmException in the try block are AmObjects, and so they all have **getName**, **getLastErrorStatus** and **clearErrorCodes** methods.

Transaction support

Messages sent and received by the AMI can, optionally, be part of a transactional unit of work. A message is included in a unit of work based on the setting of the syncpoint attribute specified in the policy used on the call. The scope of the unit of work is the session handle and only one unit of work may be active at any time.

The API calls used to control the transaction depends on the type of transaction is being used.

- WebSphere MQ messages are the only resource

  A transaction is started by the first message sent or received under syncpoint control, as specified in the policy specified for the send or receive. Multiple messages can be included in the same unit of work. The transaction is committed or backed out using the **commit** or **rollback** method.
- Using WebSphere MQ as an XA transaction coordinator

  The transaction must be started explicitly using the **begin** method before the first recoverable resource (such as a relational database) is changed. The transaction is committed or backed out using an **commit** or **rollback** method.
- Using an external transaction coordinator

  The transaction is controlled using the API calls of an external transaction coordinator (such as CICS, Encina or Tuxedo). The AMI calls are not used but the syncpoint attributed must still be specified in the policy used on the call.

Sending group messages

The AMI allows a sequence of related messages to be included in, and sent as, a message group. Group context information is sent with each message to allow the message sequence to be preserved and made available to a receiving application. To include messages in a group, the group status information of the first and subsequent messages in the group must be set as follows:

```
AMGRP_FIRST_MSG_IN_GROUP for the first message
AMGRP_MIDDLE_MSG_IN_GROUP for all messages other than first and last
AMGRP_LAST_MSG_IN_GROUP for the last message
```

The message status is set using the **AmMessage.setGroupStatus** method.

For a complete description of group messages, refer to “Sending group messages” on page 27.

Other considerations

You should also consider the following.

Multithreading

If you are using multithreading with the AMI, a session normally remains locked for the duration of a single AMI call. If you use receive with wait, the session remains locked for the duration of the wait, which might be unlimited (that is, until the wait time is exceeded or a message arrives on the queue). If you want another thread to run while a thread is waiting for a message, it must use a separate session.

AMI handles and object references can be used on a different thread from that on which they were first created for operations that do not involve an access to the underlying (WebSphere MQ) message transport. Functions such as initialize, terminate, open, close, send, receive, publish, subscribe, unsubscribe, and receive publication will access the underlying transport restricting these to the thread on which the session was first opened (for example, using **AmSession->open**). An attempt to issue these on a different thread will cause an error to be returned by WebSphere MQ and a transport error (AMRC_TRANSPORT_ERR) will be reported to the application.

Using WebSphere MQ with the AMI

You must not mix WebSphere MQ function calls with AMI calls within the same process.

Field limits

When string and binary properties such as queue name, message format, and correlation ID are set, the maximum length values are determined by WebSphere MQ, the underlying message transport. See the rules for naming WebSphere MQ objects in the *WebSphere MQ Application Programming Guide*.

Building C++ applications

This section contains information that will help you write, prepare, and run your C++ application programs on the various operating systems supported by the AMI.

AMI include files

AMI provides include files, **amtc.h** and **amtcpp.hpp**, to assist you with the writing of your applications. It is recommended that you become familiar with the contents of these files.

The include files are installed under:

```
QMQMAMI/H       (iSeries)

/amt/inc        (UNIX)

\amt\include    (Windows)
```

See “Directory structure” on page 445 (AIX), page 449 (HP-UX), page 453 (iSeries), page 458 (Solaris), or page 462 (Windows).

Your AMI C++ program must contain the statement:

```
#include <amtcpp.hpp>
```

Even though you need mention only the C++ include file, both **amtc.h** and **amtcpp.hpp** must be accessible to your program at compilation time.

Next step

Now go to one of the following to continue building a C++ application:

- “C++ applications on AIX”
- “C++ applications on HP-UX” on page 178
- “C++ applications on iSeries” on page 179
- “C++ applications on Solaris” on page 180
- “C++ applications on Windows” on page 181

C++ applications on AIX

This section explains what you have to do to prepare and run your C++ programs on the AIX operating system. See *WebSphere MQ for AIX, V5.3 Quick Beginnings* for the compilers supported by the AMI.

Preparing C++ programs on AIX

The following information is not prescriptive, because there are many ways to set up environments to build executables. Use it as a guideline, but follow your local procedures.

To compile an AMI program in a single step using the **xlC** command, you need to specify a number of options:

- Where the AMI include files are.

  To do this, use the `-I` flag. In the case of AIX, they are usually located at `/usr/mqm/amt/inc`.

- Where the AMI library is.

  To do this, use the `-L` flag. In the case of AIX, it is usually located at `/usr/mqm/lib`.
- Link with the AMI library.

  To do this, use the `-l` flag, more specifically `-lamtCpp`.

For example, to compile the C++ program `mine.cpp` into an executable called `mine`:

```
xlC -I/usr/mqm/amt/inc -L/usr/mqm/lib -lamtCpp mine.cpp -o mine
```

If, however, you are building a threaded program, you must use the correct compiler and the threaded library `libamtCpp_r.a`. For example:

```
xlC_r -I/usr/mqm/amt/inc -L/usr/mqm/lib -lamtCpp_r mine.cpp -o mine
```

Running C++ programs on AIX

To run a C++ executable, you must have access to the C++ library `libamtCpp.a` in your runtime environment. If the **amtInstall** utility has been run, this environment will be set up for you (see "Installation on AIX" on page 444).

If you have not run the utility, the easiest way of achieving this is to construct a link from the AIX default library location to the actual location of the C++ library. To do this:

```
ln -s /usr/mqm/lib/libamtCpp.a /usr/lib/libamtCpp.a
```

If you are using the threaded libraries, you can perform a similar operation:

```
ln -s /usr/mqm/lib/libamtCpp_r.a /usr/lib/libamtCpp_r.a
```

You also need access to the C libraries and WebSphere MQ in your runtime environment. To do this, make the AMI WebSphere MQ runtime binding stubs available, to allow AMI to load WebSphere MQ libraries dynamically. For the non-threaded WebSphere MQ Server library, perform:

```
ln -s /usr/mqm/lib/amtcmqm /usr/lib/amtcmqm
```

For the non-threaded WebSphere MQ Client library, perform:

```
ln -s /usr/mqm/lib/amtcmqic /usr/lib/amtcmqic
```

For the threaded WebSphere MQ Server library, perform:

```
ln -s /usr/mqm/lib/amtcmqm_r /usr/lib/amtcmqm_r
```

For the threaded WebSphere MQ Client library, perform:

```
ln -s /usr/mqm/lib/amtcmqic_r /usr/lib/amtcmqic_r
```

C++ applications on HP-UX

This section explains what you have to do to prepare and run your C++ programs on the HP-UX operating system. See *WebSphere MQ for HP-UX, V5.3 Quick Beginnings* for the compilers supported by the AMI.

Preparing C++ programs on HP-UX

The following information is not prescriptive, because there are many ways to set up environments to build executables. Use it as a guideline, but follow your local procedures.

To compile an AMI program in a single step using the **aCC** command, you need to specify a number of options:

1. Where the AMI include files are.

To do this, use the `-I` flag. In the case of HP-UX, they are usually located at `/opt/mqm/amt/inc`.

2. Where the AMI libraries are.

   To do this, use the `-Wl,+b,:,-L` flags. In the case of HP-UX, they are usually located at `/opt/mqm/lib`.
3. Link with the AMI library for C++.

   To do this, use the `-l` flag, more specifically `-lamtCpp`.

For example, to compile the C++ program `mine.cpp` into an executable called `mine`:

```
aCC +DAportable -Wl,+b,:,-L/opt/mqm/lib -o mine mine.cpp
     -I/opt/mqm/amt/inc -lamtCpp
```

Note that you could equally link to the threaded library using `-lamtCpp_r`. On HP-UX there is no difference, because the unthreaded versions of the AMI binaries are simply links to the threaded versions.

Running C++ programs on HP-UX

To run a C++ executable, you must have access to the C++ library `libamtCpp.sl` in your runtime environment. If **amtInstall** utility has been run, this environment will be set up for you (see "Installation on HP-UX" on page 448).

If you have not run the utility, the easiest way of achieving this is to construct a link from the HP-UX default library location to the actual location of the C++ library. To do this:

```
ln -s /opt/mqm/lib/libamtCpp_r.sl /usr/lib/libamtCpp.sl
```

If you are using the threaded libraries, you can perform a similar operation:

```
ln -s /opt/mqm/lib/libamtCpp_r.sl /usr/lib/libamtCpp_r.sl
```

You also need access to the C libraries and WebSphere MQ in your runtime environment. To do this, make the AMI WebSphere MQ runtime binding stubs available, to allow AMI to load WebSphere MQ libraries dynamically. For the non-threaded WebSphere MQ Server library, perform:

```
ln -s /opt/mqm/lib/amtcmqm_r /usr/lib/amtcmqm
```

For the non-threaded WebSphere MQ Client library, perform:

```
ln -s /opt/mqm/lib/amtcmqic_r /usr/lib/amtcmqic
```

For the threaded WebSphere MQ Server library, perform:

```
ln -s /opt/mqm/lib/amtcmqm_r /usr/lib/amtcmqm_r
```

For the threaded WebSphere MQ Client library, perform:

```
ln -s /opt/mqm/lib/amtcmqic_r /usr/lib/amtcmqic_r
```

As before, note that the unthreaded versions are simply links to the threaded versions.

C++ applications on iSeries

This section explains what you have to do to prepare and run your C++ programs on the iSeries system. See *WebSphere MQ for iSeries V5.3 Quick Beginnings* for the compilers supported by the AMI.

C++ applications on iSeries

Preparing C++ programs on iSeries

The following information is not prescriptive, because there are many ways to set up environments to build executables. Use it as a guideline, but follow your local procedures.

To compile a C++ module using the ILE compiler, you can use the OS/400 command **CRTCPPMOD**. The library QMQMAMI must be in the library list because it contains the amtcpp.hpp header file.

You must then bind the output of the compiler with the service program using the **CRTPGM** command. Specify the appropriate AMI service program in the **BDNSRVPGM** option of **CRTPGM**. For example:

```
CRTPGM PGM(pgmname) MODULE(pgmname) BNDSRVPGM(QMQMAMI/AMTCPP)
```

Alternatively, you can use the Visual Age C++ compiler to create your program.

Running C++ programs on iSeries

When you create your program as described in the previous section, it is bound to the service programs it requires to run. There are no additional runtime requirements.

Alternatively, you might create your program with QMQMAMI in the library list and specify *LIBL for the BNDSRVPGM parameter of CRTPGM. At run time, QMQMAMI must be in the library list.

C++ applications on Solaris

This section explains what you have to do to prepare and run your C++ programs in the Sun Solaris operating environment. See *WebSphere MQ for Solaris, V5.3 Quick Beginnings* for the compilers supported by the AMI.

Preparing C++ programs on Solaris

The following information is not prescriptive, because there are many ways to set up environments to build executables. Use it as a guideline, but follow your local procedures.

To compile an AMI program in a single step using the **CC** command, you need to specify a number of options:

- Where the AMI include files are.

  To do this, use the `-I` flag. In the case of Solaris, they are usually located at `/opt/mqm/amt/inc`.
- Where the AMI library is.

  To do this, use the `-L` flag. In the case of Solaris, it is usually located at `/opt/mqm/lib`.
- Link with the AMI library.

  To do this, use the `-l` flag, more specifically `-lamtCpp`.

For example, to compile the C++ program `mine.cpp` into an executable called `mine`:

```
CC -mt -I/opt/mqm/amt/inc -L/opt/mqm/lib -lamtCpp mine.cpp -o mine
```

Running C++ programs on Solaris

To run a C++ executable, you must have access to the C++ library `libamtCpp.so` in your runtime environment. If the **amtInstall** utility has been run, this environment will be set up for you (see “Installation on Solaris” on page 457).

If you have not run the utility, the easiest way to set up the required access is to construct a link from the Solaris default library location to the actual location of the C++ libraries.

To do this, enter:

```
ln -s /opt/mqm/lib/libamtCpp.so /usr/lib/libamtCpp.so
```

You also need access to the C libraries and WebSphere MQ in your runtime environment. To do this, make the AMI WebSphere MQ runtime binding stubs available, to allow AMI to load WebSphere MQ libraries dynamically. For the WebSphere MQ Server library, perform:

```
ln -s /opt/mqm/lib/amtcmqm /usr/lib/amtcmqm
```

For the WebSphere MQ Client library, perform:

```
ln -s /opt/mqm/lib/amtcmqic /usr/lib/amtcmqic
```

C++ applications on Windows

This section explains what you have to do to prepare and run your C++ programs on the Windows 98, Windows NT, Windows Me, and Windows 2000 operating systems. See *WebSphere MQ for Windows, V5.3 Quick Beginnings* for the compilers supported by the AMI.

Preparing C++ programs on Windows

The following information is not prescriptive, because there are many ways to set up environments to build executables. Use it as a guideline, but follow your local procedures.

To compile an AMI program in a single step using the **cl** command, you need to specify a number of options:

1. Where the AMI include files are.

   To do this, use the `/I` flag. In the case of Windows, they are usually located at `\amt\include` relative to where you installed WebSphere MQ. Alternatively, the include files could exist in one of the directories pointed to by the INCLUDE environment variable.
2. Where the AMI library is.

   To do this, include the AMT library file `amtCpp.LIB` as a command line argument. The `amtCpp.LIB` file should exist in one of the directories pointed to by the LIB environment variable.

For example, to compile the C++ program `mine.cpp` into an executable called `mine.exe`:

```
cl -IC:\MQSeries\amt\include /Fomine mine.cpp amtCpp.LIB
```

Running C++ programs on Windows

To run a C++ executable, you must have access to the C++ DLL `amtCpp.dll` in your runtime environment. Make sure it exists in one of the directories pointed to by the PATH environment variable. For example:

```
SET PATH=%PATH%;C:\MQSeries\bin;
```

If you already have WebSphere MQ installed, and you have installed AMI under the WebSphere MQ directory structure, it is likely that the PATH has already been set up for you.

C++ applications on Windows

You also need access to the C libraries and WebSphere MQ in your runtime environment. (This will be the case if you installed WebSphere MQ using the documented method.)

Chapter 8. C++ interface overview

This chapter contains an overview of the structure of the Application Messaging Interface for C++. Use it to find out what functions are available in this interface.

The C++ interface provides sets of methods for each of the classes in the following lists. The methods available for each class are listed in the following pages. Follow the page references to see the reference information for each method.

Base classes

Helper classes

Exception classes

AmSessionFactory

The **AmSessionFactory** class is used to create AmSession objects.

Constructor

Constructor for AmSessionFactory.

AmSessionFactory	page 200

Session factory management

Methods to return the name of an AmSessionFactory object, to get and set the names of the AMI data files (local host and repository), and to control traces.

getFactoryName	page 200
getLocalHost	page 200
getRepository	page 200
getTraceLevel	page 200
getTraceLocation	page 200
setLocalHost	page 201
setRepository	page 201
setTraceLevel	page 201
setTraceLocation	page 201

Create and delete session

Methods to create and delete an AmSession object.

createSession	page 200
deleteSession	page 200

AmSession

The **AmSession** object creates and manages all other objects, and provides scope for a unit of work.

Session management

Methods to open and close an AmSession object, to return its name, and to control traces.

open	page 205
close	page 202
getName	page 205
getTraceLevel	page 205
getTraceLocation	page 205

Create objects

Methods to create AmMessage, AmSender, AmReceiver, AmDistributionList, AmPublisher, AmSubscriber, and AmPolicy objects.

createMessage	page 203
createSender	page 203
createReceiver	page 203
createDistributionList	page 202
createPublisher	page 203
createSubscriber	page 203
createPolicy	page 203

Delete objects

Methods to delete AmMessage, AmSender, AmReceiver, AmDistributionList, AmPublisher, AmSubscriber, and AmPolicy objects.

deleteMessage	page 204
deleteSender	page 204
deleteReceiver	page 204
deleteDistributionList	page 204
deletePublisher	page 204
deleteSubscriber	page 204
deletePolicy	page 204

Transactional processing

Methods to begin, commit and rollback a unit of work.

begin	page 202
commit	page 202
rollback	page 205

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 202
enableWarnings	page 204
getLastErrorStatus	page 205

AmMessage

An **AmMessage** object encapsulates a WebSphere MQ message descriptor (MQMD) structure, and contains the message data.

Get values

Methods to get the coded character set ID, correlation ID, encoding, format, group status, message ID and name of the message object.

getCCSID	page 208
getCorrelationId	page 208
getElementCCSID	page 209
getEncoding	page 209
getFormat	page 209
getGroupStatus	page 210
getMessageId	page 210
getName	page 210
getReportCode	page 211
getType	page 211

Set values

Methods to set the coded character set ID, correlation ID, encoding, format, group status, feedback code type, and message type of the message object.

setCCSID	page 212
setCorrelationId	page 212
setElementCCSID	page 212
setEncoding	page 212
setFormat	page 213
setGroupStatus	page 213
setReportCode	page 213
setType	page 214

Reset values

Method to reset the message object to the state it had when first created.

reset	page 211

Read and write data

Methods to read or write byte data to or from the message object, to get and set the data offset, and to get the length of the data.

getDataLength	page 208
getDataOffset	page 208
setDataOffset	page 212
readBytes	page 211

writeBytes	page 214

Publish/subscribe topics

Methods to manipulate the topics in a publish/subscribe message.

addTopic	page 207
deleteTopic	page 208
getTopic	page 211
getTopicCount	page 211

Publish/subscribe filters

Methods to manipulate filters for content-based publish/subscribe.

addFilter	page 207
deleteFilter	page 207
getFilter	page 209
getFilterCount	page 209

Publish/subscribe name/value elements

Methods to manipulate the name/value elements in a publish/subscribe message.

addElement	page 207
deleteElement	page 207
getElement	page 209
getElementCount	page 209
deleteNamedElement	page 208
getNamedElement	page 210
getNamedElementCount	page 210

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 207
enableWarnings	page 208
getLastErrorStatus	page 210

AmSender

An **AmSender** object encapsulates a WebSphere MQ object descriptor (MQOD) structure.

Open and close

Methods to open and close the sender service.

open	page 216
close	page 215

Send

Method to send a message.

send	page 216

Send file

Method to send data from a file

sendFile	page 217

Get values

Methods to get the coded character set ID, encoding and name of the sender service.

getCCSID	page 215
getEncoding	page 215
getName	page 216

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 215
enableWarnings	page 215
getLastErrorStatus	page 216

AmReceiver

An **AmReceiver** object encapsulates a WebSphere MQ object descriptor (MQOD) structure.

Open and close

Methods to open and close the receiver service.

open	page 220
close	page 219

Receive and browse

Methods to receive or browse a message.

receive	page 220
browse	page 218

Receive file

Method to receive file message data into a file.

receiveFile	page 221

Get values

Methods to get the definition type, name and queue name of the receiver service.

getDefinitionType	page 219
getName	page 220
getQueueName	page 220

Set value

Method to set the queue name of the receiver service.

setQueueName	page 221

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 219
enableWarnings	page 219
getLastErrorStatus	page 220

AmDistributionList

An **AmDistributionList** object encapsulates a list of AmSender objects.

Open and close

Methods to open and close the distribution list service.

open	page 222
close	page 222

Send

Method to send a message to the distribution list.

send	page 223

Send file

Method to send date from a file to the each sender defined in the distribution list.

sendFile	page 223

Get values

Methods to get the name of the distribution list service, a count of the AmSenders in the list, and one of the AmSenders that is contained in the list.

getName	page 222
getSenderCount	page 222
getSender	page 222

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 222
enableWarnings	page 222
getLastErrorStatus	page 222

AmPublisher

An **AmPublisher** object encapsulates a sender service and provides support for publishing messages to a publish/subscribe broker.

Open and close

Methods to open and close the publisher service.

open	page 225
close	page 224

Publish

Method to publish a message.

publish	page 225

Get values

Methods to get the coded character set ID, encoding and name of the publisher service.

getCCSID	page 224
getEncoding	page 224
getName	page 224

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 224
enableWarnings	page 224
getLastErrorStatus	page 224

AmSubscriber

An **AmSubscriber** object encapsulates both a sender service and a receiver service. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

Open and close

Methods to open and close the subscriber service.

open	page 227
close	page 226

Broker messages

Methods to subscribe to a broker, remove a subscription, and receive a publication from the broker.

subscribe	page 228
unsubscribe	page 228
receive	page 227

Get values

Methods to get the coded character set ID, definition type, encoding, name and queue name of the subscriber service.

getCCSID	page 226
getDefinitionType	page 226
getEncoding	page 226
getName	page 227
getQueueName	page 227

Set value

Method to set the queue name of the subscriber service.

setQueueName	page 227

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 226
enableWarnings	page 226
getLastErrorStatus	page 226

AmPolicy

An **AmPolicy** object encapsulates the options used during AMI operations.

Policy management

Methods to return the name of the policy, and to get and set the wait time when receiving a message.

getName	page 229
getWaitTime	page 229
setWaitTime	page 229

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 229
enableWarnings	page 229
getLastErrorStatus	page 229

Helper classes

The classes that encapsulate name/value elements for publish/subscribe, strings, binary data and error status.

AmBytes

The AmBytes class is an encapsulation of a byte array. It allows the AMI to pass byte strings across the interface and enables manipulation of byte strings. It contains constructors, operators and a destructor, and methods to copy, compare, and pad. AmBytes also has methods to give the length of the encapsulated bytes and a method to reference the data contained within an AmBytes object.

constructors	page 230
destructor	page 231
operators	page 231
cmp	page 230
cpy	page 231
dataPtr	page 231
length	page 231
pad	page 231

AmElement

Constructor for AmElement, and methods to return the name, type, value and version of an element, to set the version, and to return an AmString representation of the element.

AmElement	page 232
getName	page 232
getValue	page 232
getVersion	page 232
setVersion	page 232
toString	page 232

AmObject

A virtual class containing methods to return the name of the object, to clear the error codes and to return the last error condition.

clearErrorCodes	page 233
getLastErrorStatus	page 233
getName	page 233

AmStatus

Constructor for AmStatus, and methods to return the completion code, reason code, secondary reason code and status text, and to return an AmString representation of the AmStatus.

AmStatus	page 234
getCompletionCode	page 234

getReasonCode	page 234
getReasonCode2	page 234
toString	page 234

AmString

The AmString class is an encapsulation of a string. It allows the AMI to pass strings across the interface and enables manipulation of strings. It contains constructors, operators, a destructor, and methods to copy, concatenate, pad, split, truncate and strip. AmString also has methods to give the length of the encapsulated string, compare AmStrings, check whether one AmString is contained within another and a method to reference the text of an AmString.

constructors	page 235
destructor	page 235
operators	page 236
cat	page 235
cmp	page 235
contains	page 235
cpy	page 235
length	page 236
pad	page 236
split	page 236
strip	page 236
text	page 236
truncate	page 236

Exception classes

Classes that encapsulate error and warning conditions. AmErrorException and AmWarningException inherit from AmException.

AmException

Methods to return the completion code and reason code from the Exception, the class name, method name and source of the Exception, and to return a string representation of the Exception.

getClassName	page 237
getCompletionCode	page 237
getMethodName	page 237
getReasonCode	page 237
getSource	page 237
toString	page 237

AmErrorException

Methods to return the completion code and reason code from the Exception, the class name, method name and source of the Exception, and to return a string representation of the Exception.

getClassName	page 238
getCompletionCode	page 238
getMethodName	page 238
getReasonCode	page 238
getSource	page 238
toString	page 238

AmWarningException

Methods to return the completion code and reason code from the Exception, the class name, method name and source of the Exception, and to return a string representation of the Exception.

getClassName	page 239
getCompletionCode	page 239
getMethodName	page 239
getReasonCode	page 239
getSource	page 239
toString	page 239

C++ interface overview

Chapter 9. C++ interface reference

In the following sections the C++ interface methods are listed by the class they refer to. Within each section the methods are listed in alphabetical order.

Base classes

Note that all of the methods in these classes can throw AmWarningException and AmErrorException (see below). However, by default, AmWarningExceptions are not raised.

Helper classes

Exception classes

AmSessionFactory

The **AmSessionFactory** class is used to create AmSession objects.

AmSessionFactory

Constructors for an AmSessionFactory.

```
AmSessionFactory();
AmSessionFactory(char * name);
```

name	The name of the AmSessionFactory. This is the location of the data files used by the AMI (the repository file and the local host file). The name should be a fully qualified directory that includes the path under which the files are located. Otherwise, see "Local host and repository files (iSeries, UNIX, and Windows)" on page 468 for the location of these files.

createSession

Creates an AmSession object.

```
AmSession * createSession(char * name);
```

name	The name of the AmSession.

deleteSession

Deletes an AmSession object previously created using the **createSession** method.

```
void  deleteSession(AmSession ** pSession);
```

pSession	A pointer to the AmSession pointer returned by the **createSession** method.

getFactoryName

Returns the name of the AmSessionFactory.

```
AmString  getFactoryName();
```

getLocalHost

Returns the name of the local host file.

```
AmString  getLocalHost();
```

getRepository

Returns the name of the repository file.

```
AmString  getRepository();
```

getTraceLevel

Returns the trace level for the AmSessionFactory.

```
int  getTraceLevel();
```

getTraceLocation

Returns the location of the trace for the AmSessionFactory.

```
AmString  getTraceLocation();
```

setLocalHost

Sets the name of the AMI local host file to be used by any AmSession created from this AmSessionFactory. (Otherwise, the default host file `amthost.xml` is used.)

```
void  setLocalHost(char * fileName);
```

fileName	The name of the file used by the AMI as the local host file. This file must be present on the local file system or an error will be produced upon the creation of an AmSession.

setRepository

Sets the name of the AMI repository to be used by any AmSession created from this AmSessionFactory. (Otherwise, the default repository file `amt.xml` is used.)

```
void  setRepository(char * fileName);
```

fileName Either of the following:

- The name of the file used by the AMI as the repository.

  This file must be present on the local file system or an error will be produced upon the creation of an AmSession.
- A reference to the repository information in LDAP URL format, when repository information is obtained from an LDAP directory.

  For details about specifying an LDAP URL, see "Directory search" on page 512.

setTraceLevel

Sets the trace level for the AmSessionFactory.

```
void  setTraceLevel(int level);
```

level	The trace level to be set in the AmSessionFactory. Trace levels are 0 through 9, where 0 represents minimal tracing and 9 represents a fully detailed trace.

setTraceLocation

Sets the location of the trace for the AmSessionFactory.

```
void  setTraceLocation(char * location);
```

location	The location on the local system where trace files will be written. This location must be a directory, and it must exist before the trace is run.

AmSession

An **AmSession** object provides the scope for a unit of work and creates and manages all other objects, including at least one connection object. Each (WebSphere MQ) connection object encapsulates a single WebSphere MQ queue manager connection. The session object definition specifying the required set of queue manager connection(s) can be provided by a repository policy definition, or by default will name a single local queue manager with no repository. The session, when deleted, is responsible for releasing memory by closing and deleting all other objects that it manages.

Note that you should not mix WebSphere MQ MQCONN or MQDISC requests (or their equivalent in the WebSphere MQ C++ interface) on the same thread as AMI calls, otherwise premature disconnection might occur.

begin

Begins a unit of work in this AmSession, allowing an AMI application to take advantage of the resource coordination provided in WebSphere MQ. The unit of work can subsequently be committed by the **commit** method, or backed out by the **rollback** method. This should be used only when AMI is the transaction coordinator. If available, native coordination APIs (for example CICS or Tuxedo) should be used.

begin is overloaded. The `policy` parameter is optional.

```
void  begin(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

clearErrorCodes

Clears the error codes in the AmSession.

```
void  clearErrorCodes();
```

close

Closes the AmSession, and all open objects owned by it. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

commit

Commits a unit of work that was started by **AmSession.begin**. **commit** is overloaded: the `policy` parameter is optional.

```
void  commit(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

createDistributionList

Creates an AmDistributionList object.

```
AmDistributionList * createDistributionList(char * name);
```

`name`	The name of the AmDistributionList. This must match the name of a distribution list defined in the repository.

createMessage

Creates an AmMessage object.

```
AmMessage * createMessage(char * name);
```

name	The name of the AmMessage. This can be any name that is meaningful to the application.

createPolicy

Creates an AmPolicy object.

```
AmPolicy * createPolicy(char * name);
```

name	The name of the AmPolicy. If it matches a policy defined in the repository, the policy will be created using the repository definition, otherwise it will be created with default values.

createPublisher

Creates an AmPublisher object.

```
AmPublisher * createPublisher(char * name);
```

name	The name of the AmPublisher. If it matches a publisher defined in the repository, the publisher will be created using the repository definition, otherwise it will be created with default values (that is, with an AmSender name that matches the publisher name).

createReceiver

Creates an AmReceiver object.

```
AmReceiver * createReceiver(char * name);
```

name	The name of the AmReceiver. If it matches a receiver defined in the repository, the receiver will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the receiver name).

createSender

Creates an AmSender object.

```
AmSender * createSender(char * name);
```

name	The name of the AmSender. If it matches a sender defined in the repository, the sender will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the sender name).

createSubscriber

Creates an AmSubscriber object.

```
AmSubscriber * createSubscriber(char * name);
```

name	The name of the AmSubscriber. If it matches a subscriber defined in the repository, the subscriber will be created using the repository definition, otherwise it will be created with default values (that is, with an AmSender name that matches the subscriber name, and an AmReceiver name that is the same with the addition of the suffix '.RECEIVER').

deleteDistributionList

Deletes an AmDistributionList object.

```
void  deleteDistributionList(AmDistributionList ** dList);
```

dList	A pointer to the AmDistributionList * returned on a createDistributionList call.

deleteMessage

Deletes an AmMessage object.

```
void  deleteMessage(AmMessage ** message);
```

message	A pointer to the AmMessage * returned on a createMessage call.

deletePolicy

Deletes an AmPolicy object.

```
void  deletePolicy(AmPolicy ** policy);
```

policy	A pointer to the AmPolicy * returned on a createPolicy call.

deletePublisher

Deletes an AmPublisher object.

```
void  deletePublisher(AmPublisher ** publisher);
```

publisher	A pointer to the AmPublisher returned on a createPublisher call.

deleteReceiver

Deletes an AmReceiver object.

```
void  deleteReceiver(AmReceiver ** receiver);
```

receiver	A pointer to the AmReceiver returned on a createReceiver call.

deleteSender

Deletes an AmSender object.

```
void  deleteSender(AmSender ** sender);
```

sender	A pointer to the AmSender returned on a createSender call.

deleteSubscriber

Deletes an AmSubscriber object.

```
void  deleteSubscriber(AmSubscriber ** subscriber);
```

subscriber	A pointer to the AmSubscriber returned on a createSubscriber call.

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(AMBOOL warningsOn);
```

warningsOn	If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmSession.

```
String  getName();
```

getTraceLevel

Returns the trace level of the AmSession.

```
int  getTraceLevel();
```

getTraceLocation

Returns the location of the trace for the AmSession.

```
AmString  getTraceLocation();
```

open

Opens an AmSession using the specified policy. The application profile group of this policy provides the connection definitions enabling the connection objects to be created. The specified library is loaded for each connection and its dispatch table initialized. If the transport type is WebSphere MQ and the WebSphere MQ local queue manager library cannot be loaded, the WebSphere MQ client queue manager is loaded. Each connection object is then opened.

open is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

rollback

Rolls back a unit of work that was started by **AmSession.begin**, or under policy control. **rollback** is overloaded: the `policy` parameter is optional.

```
void  rollback(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmMessage

An **AmMessage** object encapsulates the WebSphere MQ MQMD message properties, and name/value elements such as the topics for publish/subscribe messages. In addition it contains the application data.

The initial state of the message object is:

CCSID	default queue manager CCSID
correlationId	all zeros
dataLength	zero
dataOffset	zero
elementCount	zero
encoding	AMENC_NATIVE
format	AMFMT_STRING
groupStatus	AMGRP_MSG_NOT_IN_GROUP
reportCode	AMFB_NONE
topicCount	zero
type	AMMT_DATAGRAM

When a message object is used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see **reset** on page 211) and rebuild it each time.

Note that the following methods are only valid after a session has been opened with **AmSession.open** or after you have explicitly set the element CCSID with **AmMessage.setElementCCSID**:

addElement	page 207
deleteElement	page 207
getElement	page 209
getElementCount	page 209
deleteNamedElement	page 208
getNamedElement	page 210
getNamedElementCount	page 210
addTopic	page 207
deleteTopic	page 208
getTopic	page 211
getTopicCount	page 211

addElement

Adds a name/value element to an AmMessage object. **addElement** is overloaded: the `element` parameter is required, but the `options` parameter is optional.

```
void  addElement(
  AmElement  &element,
  int        options);
```

`element`	The element to be added to the AmMessage.
`options`	The options to be used. This parameter is reserved and must be set to zero.

addFilter

Adds a publish/subscribe filter to an AmMessage object.

```
void  addFilter(char * filter);
```

`filter`	The filter to be added to the AmMessage.

addTopic

Adds a publish/subscribe topic to an AmMessage object.

```
void  addTopic(char * topicName);
```

`topicName`	The name of the topic to be added to the AmMessage.

clearErrorCodes

Clears the error in the AmMessage object.

```
void  clearErrorCodes();
```

deleteElement

Deletes the element in the AmMessage object at the specified index. Indexing is within all elements of a message, and might include topics (which are specialized elements).

```
void  deleteElement(int index);
```

`index`	The index of the element to be deleted, starting from zero. On completion, elements with higher `index` values than that specified will have those values reduced by one. **getElementCount** gets the number of elements in the message.

deleteFilter

Deletes a publish/subscribe filter in an AmMessage object at the specified index. Indexing is within all filters in the message.

```
void  deleteFilter(int filterIndex);
```

`filterIndex`	The index of the filter to be deleted, starting from zero. **getFilterCount** gets the number of filters in a message.

deleteNamedElement

Deletes the element with the specified name in the AmMessage object, at the specified index. Indexing is within all elements that share the same name.

```
void  deleteNamedElement(
  char * name,
  int    index);
```

name	The name of the element to be deleted.
index	The index of the element to be deleted, starting from zero. On completion, elements with higher `index` values than that specified will have those values reduced by one. **getNamedElementCount** gets the number of elements in the message with the specified name.

deleteTopic

Deletes a publish/subscribe topic in an AmMessage object at the specified index. Indexing is within all topics in the message.

```
void  deleteTopic(int index);
```

index	The index of the topic to be deleted, starting from zero. **getTopicCount** gets the number of topics in the message.

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(AMBOOL warningsOn);
```

warningsOn	If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier used by the AmMessage.

```
int  getCCSID();
```

getCorrelationId

Returns the correlation identifier for the AmMessage.

```
AmBytes  getCorrelationId();
```

getDataLength

Returns the length of the message data in the AmMessage.

```
int  getDataLength();
```

getDataOffset

Returns the current offset in the message data for reading or writing data bytes.

```
int  getDataOffset();
```

getElement

Returns an element in an AmMessage object at the specified index. Indexing is within all elements in the message, and might include topics (which are specialized elements).

```
AmElement  getElement(int index);
```

`index`	The index of the element to be returned, starting from zero. **getElementCount** gets the number of elements in the message.

getElementCCSID

Returns the message element CCSID. This is the coded character set identifier for passing message element data (including topic and filter data) to or from an application.

```
int  getElementCCSID();
```

getElementCount

Returns the total number of elements in an AmMessage object. This might include topics (which are specialized elements).

```
int  getElementCount();
```

getEncoding

Returns the value used to encode numeric data types for the AmMessage.

```
int  getEncoding();
```

The following values can be returned:

```
AMENC_NATIVE
AMENC_NORMAL
AMENC_NORMAL_FLOAT_390
AMENC_REVERSED
AMENC_REVERSED_FLOAT_390
AMENC_UNDEFINED
```

getFilter

Returns the publish/subscribe filter in the AmMessage object at the specified index. Indexing is within all filters.

```
AmString  getFilter(int filterIndex);
```

`filterIndex`	The index of the filter to be returned, starting from zero. **getElementCount** gets the number of filters in a message.

getFilterCount

Returns the total number of publish/subscribe filters in the AmMessage object.

```
AmElement  getFilterCount();
```

getFormat

Returns the format of the AmMessage.

```
AmString  getFormat();
```

The following values can be returned:

```
AMFMT_NONE
AMFMT_STRING
AMFMT_RF_HEADER
```

getGroupStatus

Returns the group status value for the AmMessage. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group.

```
int  getGroupStatus();
```

The following values can be returned:

```
AMGRP_MSG_NOT_IN_GROUP
AMGRP_FIRST_MSG_IN_GROUP
AMGRP_MIDDLE_MSG_IN_GROUP
AMGRP_LAST_MSG_IN_GROUP
AMGRP_ONLY_MSG_IN_GROUP
```

Alternatively, bitwise tests can be performed using the constants:

```
AMGF_IN_GROUP
AMGF_FIRST
AMGF_LAST
```

getLastErrorStatus

Returns the AmStatus of the last error condition for this object.

```
AmStatus  getLastErrorStatus();
```

getMessageId

Returns the message identifier from the AmMessage object.

```
AmBytes  getMessageId();
```

getName

Returns the name of the AmMessage object.

```
AmString  getName();
```

getNamedElement

Returns the element with the specified name in an AmMessage object, at the specified index. Indexing is within all elements that share the same name.

```
AmElement  getNamedElement(
  char *  name,
  int     index);
```

name	The name of the element to be returned.
index	The index of the element to be returned, starting from zero.

getNamedElementCount

Returns the total number of elements with the specified name in the AmMessage object.

```
int  getNamedElementCount(char * name);
```

name	The name of the elements to be counted.

getReportCode

Returns the feedback code from an AmMessage of type AMMT_REPORT.

```
int  getReportCode();
```

In addition to application defined values, the following values can be returned:

```
AMFB_NONE
AMFB_CODE_EXPIRATION
AMFB_CODE_COA
AMFB_CODE_COD
```

getTopic

Returns the publish/subscribe topic in the AmMessage object, at the specified index. Indexing is within all topics.

```
AmString  getTopic(int index);
```

index	The index of the topic to be returned, starting from zero. **getTopicCount** gets the number of topics in the message.

getTopicCount

Returns the total number of publish/subscribe topics in the AmMessage object.

```
int  getTopicCount();
```

getType

Returns the message type from the AmMessage.

```
int  getType();
```

The following values can be returned:

```
AMMT_REQUEST
AMMT_REPLY
AMMT_REPORT
AMMT_DATAGRAM
```

readBytes

Populates an AmByte object with data from the AmMessage, starting at the current data offset (which must be positioned before the end of the data for the read to be successful). Use **setDataOffset** to specify the data offset. **readBytes** will advance the data offset by the number of bytes read, leaving the offset immediately after the last byte read.

```
AmBytes  readBytes(int dataLength);
```

dataLength	The maximum number of bytes to be read from the message data. The number of bytes returned is the minimum of `dataLength` and the number of bytes between the data offset and the end of the data.

reset

Resets the AmMessage object to its initial state (see page 206).

reset is overloaded: the `options` parameter is optional.

```
void  reset(int options);
```

options	A reserved field that must be set to zero.

setCCSID

Sets the coded character set identifier used by the AmMessage object.

```
void  setCCSID(int codedCharSetId);
```

codedCharSetId
The CCSID to be set in the AmMessage.

setCorrelationId

Sets the correlation identifier in the AmMessage object.

```
void  setCorrelationId(AmBytes &correlId);
```

correlId An AmBytes object containing the correlation identifier to be set in the AmMessage. The correlation identifier can be reset by specifying this as an empty AmBytes object.

setDataOffset

Sets the data offset for reading or writing byte data.

```
void  setDataOffset(int dataOffset);
```

dataOffset The data offset to be set in the AmMessage. Set an offset of zero to read or write from the start of the data.

setElementCCSID

This specifies the character set to be used for subsequent message element data (including topic and filter data) passed to or returned from the application. Existing elements in the message are unmodified (but will be returned in the character set). The default value of element CCSID is the queue manager CCSID.

```
void  setElementCCSID(int elementCCSID);
```

elementCCSID The element CCSID to be set in the AmMessage.

setEncoding

Sets the encoding of the data in the AmMessage object.

```
void  setEncoding(int encoding);
```

encoding The encoding to be used in the AmMessage. It can take one of the following values:

```
AMENC_NATIVE
AMENC_NORMAL
AMENC_NORMAL_FLOAT_390
AMENC_REVERSED
AMENC_REVERSED_FLOAT_390
AMENC_UNDEFINED
```

setFormat

Sets the format for the AmMessage object.

```
void  setFormat(char * format);
```

format	The format to be used in the AmMessage. It can take one of the following values: AMFMT_NONE AMFMT_STRING AMFMT_RF_HEADER If set to AMFMT_NONE, the default format for the sender will be used (if available). Specify as NULL to set the format to NULL.

setGroupStatus

Sets the group status value for the AmMessage. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group. Once you start sending messages in a group, you must complete the group before sending any messages that are not in the group.

If you specify AMGRP_MIDDLE_MSG_IN_GROUP or AMGRP_LAST_MSG_IN_GROUP without specifying AMGRP_FIRST_MSG_IN_GROUP, the behavior is the same as for AMGRP_FIRST_MSG_IN_GROUP and AMGRP_ONLY_MSG_IN_GROUP.

If you specify AMGRP_FIRST_MSG_IN_GROUP out of sequence, the behavior is the same as for AMGRP_MIDDLE_MSG_IN_GROUP.

```
void  setGroupStatus(int groupStatus);
```

groupStatus	The group status to be set in the AmMessage. It can take one of the following values: AMGRP_MSG_NOT_IN_GROUP AMGRP_FIRST_MSG_IN_GROUP AMGRP_MIDDLE_MSG_IN_GROUP AMGRP_LAST_MSG_IN_GROUP AMGRP_ONLY_MSG_IN_GROUP

setReportCode

Sets the feedback code used by the AmMessage object. This is meaningful only for a message of type AMMT_REPORT.

```
void setReportCode(int reportCode);
```

reportCode	The feedback (or report code) value set in the AmMessage. In addition to application defined values, the following values can be set: AMFB_NONE AMFB_CODE_EXPIRATION AMFB_CODE_COA AMFB_CODE_COD

setType

Sets the message type used by the AmMessage object. If a response message is requested with a publish, subscribe, or unsubscribe request, the specified value is ignored and message type AMMT_REQUEST is used. If the value specified is AMMT_DATAGRAM, this is overridden when requesting or sending a response message (by AMMT_REQUEST and AMMT_RESPONSE, respectively).

```
void setType(int type);
```

type	The message type to be set in the AmMessage. It can take one of the following values:
	`AMMT_DATAGRAM`
	`AMMT_REQUEST`
	`AMMT_REPLY`
	`AMMT_REPORT`

writeBytes

Writes a byte array into the AmMessage object, starting at the current data offset. If the data offset is not at the end of the data, existing data is overwritten. Use **setDataOffset** to specify the data offset. **writeBytes** will advance the data offset by the number of bytes written, leaving it immediately after the last byte written.

```
void  writeBytes(AmBytes &data);
```

data	An AmBytes object containing the data to be written to the AmMessage.

AmSender

An **AmSender** object encapsulates a WebSphere MQ object descriptor (MQOD) structure. This represents a WebSphere MQ queue on a local or remote queue manager. An open sender service is always associated with an open connection object (such as a queue manager connection). Support is also included for dynamic sender services (those that encapsulate model queues). The required sender service object definitions can be provided from a repository, or created without a repository definition by defaulting to the existing queue objects on the local queue manager.

The AmSender object must be created before it can be opened. This is done using **AmSession.createSender**.

A *responder* is a special type of AmSender used for sending a response to a request message. It is not created from a repository definition. Once created, it must not be opened until used in its correct context as a responder receiving a request message with **AmReceiver.receive**. When opened, its queue and queue manager properties are modified to reflect the `ReplyTo` destination specified in the message being received. When first used in this context, the sender service becomes a responder sender service.

clearErrorCodes

Clears the error codes in the AmSender.

```
void  clearErrorCodes();
```

close

Closes the AmSender. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(AMBOOL warningsOn);
```

`warningsOn`	If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier for the AmSender. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the sender must perform CCSID conversion of the message before it is sent.

```
int  getCCSID();
```

getEncoding

Returns the value used to encode numeric data types for the AmSender. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the sender must convert the encoding of the message before it is sent.

```
int  getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmSender.

```
AmString  getName();
```

open

Opens an AmSender service. **open** is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

send

Sends a message using the AmSender service. If the AmSender is not open, it will be opened (if this action is specified in the policy options).

send is overloaded: the `sendMessage` parameter is required, but the others are optional. `receivedMessage` and `responseService` are used in request/response messaging, and are mutually exclusive.

```
void  send(
  AmMessage   &sendMessage,
  AmReceiver  &responseService,
  AmMessage   &receivedMessage,
  AmPolicy    &policy);
```

`sendMessage`	The message object that contains the data to be sent.
`responseService`	The AmReceiver to which the response to this message should be sent. Omit it if no response is required.
`receivedMessage`	The previously received message which is used for correlation with the sent message. If omitted, the sent message is not correlated with any received message.
`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

sendFile

Sends data from a file. To send data from a file, the sendMessage and fileName parameters are required, but the policy is optional. The file data can be received as normal message data by a target application using AmReceiver.receive, or used to reconstruct the file with AmReceiver.receiveFile.

```
void sendFile(
  AmMessage &sendMessage,
  char *    filename,
  AmPolicy  &policy);
```

sendMessage	The message object to use to send the file. This can be used to specify the Correlation ID for example.
fileName	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with the receive file method (see “receiveFile” on page 221 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
policy	The policy to be used. If omitted, the system default policy (name constant : AMSD_POL) is used.

AmReceiver

An **AmReceiver** object encapsulates a WebSphere MQ object descriptor (MQOD) structure. This represents a WebSphere MQ queue on a local or remote queue manager. An open AmReceiver is always associated with an open connection object, such as a queue manager connection. Support is also included for a dynamic AmReceiver (that encapsulates a model queue). The required AmReceiver object definitions can be provided from a repository or can be created automatically from the set of existing queue objects available on the local queue manager.

There is a definition type associated with each AmReceiver:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

An AmReceiver created from a repository definition will be initially of type AMDT_PREDEFINED or AMDT_DYNAMIC. When opened, its definition type might change from AMDT_DYNAMIC to AMDT_TEMP_DYNAMIC according to the properties of its underlying queue object.

An AmReceiver created with default values (that is, without a repository definition) will have its definition type set to AMDT_UNDEFINED until it is opened. When opened, this will become AMDT_DYNAMIC, AMDT_TEMP_DYNAMIC, or AMDT_PREDEFINED, according to the properties of its underlying queue object.

browse

Browses an AmReceiver service. **browse** is overloaded: the `browseMessage` and `options` parameters are required, but the others are optional.

```
void  browse(
  AmMessage   &browseMessage,
  int         options,
  AmSender    &responseService,
  AmMessage   &selectionMessage,
  AmPolicy    &policy);
```

browseMessage The message object that receives the browse data.

options Options controlling the browse operation. Possible values are:

```
AMBRW_NEXT
AMBRW_FIRST
AMBRW_CURRENT
AMBRW_RECEIVE_CURRENT
AMBRW_DEFAULT          (AMBRW_NEXT)
AMBRW_LOCK_NEXT        (AMBRW_LOCK + AMBRW_NEXT)
AMBRW_LOCK_FIRST       (AMBRW_LOCK + AMBRW_FIRST)
AMBRW_LOCK_CURRENT     (AMBRW_LOCK + AMBRW_CURRENT)
AMBRW_UNLOCK
```

`AMBRW_RECEIVE_CURRENT` is equivalent to **AmReceiver.receive** for the message under the browse cursor.

Note that a locked message is unlocked by another browse or receive, even though it is not for the same message.

`responseService`	The AmSender to be used for sending any response to the browsed message. If omitted, no response can be sent. Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.
`selectionMessage`	A message object which contains the Correlation ID used to selectively browse a message from the AmReceiver. If omitted, the first available message is browsed. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be be reset (see **AmMessage.setCorrelationId** on page 212) before invoking the browse method.
`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

clearErrorCodes

Clears the error codes in the AmReceiver.

```
void  clearErrorCodes();
```

close

Closes the AmReceiver. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(AMBOOL warningsOn);
```

`warningsOn`	If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getDefinitionType

Returns the definition type (service type) for the AmReceiver.

```
int  getDefinitionType();
```

The following values can be returned:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

Values other than AMDT_UNDEFINED reflect the properties of the underlying queue object.

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmReceiver.

```
AmString  getName();
```

getQueueName

Returns the queue name of the AmReceiver. This is used to determine the queue name of a permanent dynamic AmReceiver, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **setQueueName**.)

```
AmString  getQueueName();
```

open

Opens an AmReceiver service. **open** is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy &policy);
```

`policy` The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

receive

Receives a message from the AmReceiver service. **receive** is overloaded: the `receiveMessage` parameter is required, but the others are optional.

```
void  receive(
  AmMessage   &receiveMessage,
  AmSender    &responseService,
  AmMessage   &selectionMessage,
  AmPolicy    &policy);
```

`receiveMessage`
The message object that receives the data. The message object is reset implicitly before the receive takes place.

`responseService`
The AmSender to be used for sending any response to the received message. If omitted, no response can be sent.

`selectionMessage`
A message object containing the Correlation ID used to selectively receive a message from the AmReceiver. If omitted, the first available message is received. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be be reset (see **AmMessage.setCorrelationId** on page 212) before invoking the receive method.

`policy` The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

receiveFile

Receives file message data into a file. To receive data into a file, the receiveMessage parameter is required, but the others are optional.

```
void receiveFile(
  AmMessage &receiveMessage,
  char *    &fileName,
  AmMessage &selectionMessage,
  AmPolicy  &policy);
```

receiveMessage	The message object used to receive the file. This is updated with the message properties, for example the Message ID. If the message is not from a file, the message object receives the data. The message object is reset implicitly before the receive takes place.
fileName	The name of the file to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If NULL or a null string is specified, the AMI will use the name of the originating file (including any directory prefix), exactly as it was supplied on the send file call. Note that the original file name may not be appropriate for use by the receiver, either because a path name included in the file name is not applicable to the receiving system, or because the sending and receiving systems use different file naming conventions.
selectionMessage	A message object containing the Correlation ID used to selectively receive a message from the AmReceiver. If omitted, the first available message is received. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be reset (see **AmMessage.setCorrelationId** on page 212) before invoking the receive method.
policy	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

setQueueName

Sets the queue name of the AmReceiver (when this encapsulates a model queue). This is used to specify the queue name of a recreated permanent dynamic AmReceiver, in order to receive messages in a session subsequent to the one in which it was created. (See also **getQueueName**.)

```
void  setQueueName(char * queueName);
```

queueName	The queue name to be set in the AmReceiver.

AmDistributionList

An **AmDistributionList** object encapsulates a list of AmSender objects.

clearErrorCodes

Clears the error codes in the AmDistributionList.

```
void  clearErrorCodes();
```

close

Closes the AmDistributionList. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(AMBOOL warningsOn);
```

`warningsOn`	If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getLastErrorStatus

Returns the AmStatus of the last error condition of this object.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmDistributionList object.

```
AmString  getName();
```

getSender

Returns a pointer to the AmSender object contained within the AmDistributionList object at the index specified. `AmDistributionList.getSenderCount` gets the number of AmSender services in the distribution list.

```
AmSender * getSender(int index);
```

`index`	The index of the AmSender in the AmDistributionList, starting at zero.

getSenderCount

Returns the number of AmSender services in the AmDistributionList object.

```
int  getSenderCount();
```

open

Opens an AmDistributionList object for each of the destinations in the distribution list. **open** is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

send

Sends a message to each AmSender defined in the AmDistributionList object. **send** is overloaded: the `sendMessage` parameter is required, but the others are optional.

```
void  send(
  AmMessage   &sendMessage,
  AmReceiver  &responseService,
  AmPolicy    &policy);
```

`sendMessage`	The message object containing the data to be sent.
`responseService`	The AmReceiver to be used for receiving any response to the sent message. If omitted, no response can be received.
`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

sendFile

Sends data from a file to each AmSender defined in the AmDistributionList object. The `sendMessage` and `fileName` parameters are required to send data from a file, but the policy is optional. The file data can be received as normal message data by a target application using AmReceiver.receive, or used to reconstruct the file with AmReceiver.receiveFile.

```
void  sendFile(
  AmMessage   &sendMessage,
  char*       fileName,
  AmPolicy    &policy);
```

`sendMessage`	The message object to use to send the file. This can be used to specify the Correlation ID, for example. The message must not include any elements or data.
`fileName`	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with the receive file method (see "receiveFile" on page 221 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
`policy`	The policy to be used. If omitted, the system default policy (name constant: AMSD_POL) is used.

AmPublisher

An **AmPublisher** object encapsulates an AmSender and provides support for publish requests to a publish/subscribe broker.

clearErrorCodes

Clears the error codes in the AmPublisher.

```
void  clearErrorCodes();
```

close

Closes the AmPublisher. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(AMBOOL warningsOn);
```

`warningsOn`	If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier for the AmPublisher. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the publisher must perform CCSID conversion of the message before it is sent.

```
int  getCCSID();
```

getEncoding

Returns the value used to encode numeric data types for the AmPublisher. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the publisher must convert the encoding of the message before it is sent.

```
int  getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmPublisher.

```
AmString  getName();
```

open

Opens an AmPublisher service. **open** is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

publish

Publishes a message using the AmPublisher. **publish** is overloaded: the `pubMessage` parameter is required, but the others are optional.

```
void  publish(
  AmMessage   &pubMessage,
  AmReceiver  &responseService,
  AmPolicy    &policy);
```

`pubMessage`	The message object that contains the data to be published.
`responseService`	The AmReceiver to which the response to this publish request should be sent. Omit it if no response is required. This parameter is mandatory if the policy specifies implicit registration of the publisher.
`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmSubscriber

An **AmSubscriber** object encapsulates both an AmSender and an AmReceiver. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

clearErrorCodes

Clears the error codes in the AmSubscriber.

```
void  clearErrorCodes();
```

close

Closes the AmSubscriber. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy &policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(AMBOOL warningsOn);
```

`warningsOn`	If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier for the AmSender in the AmSubscriber. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the subscriber must perform CCSID conversion of the message before it is sent.

```
int  getCCSID();
```

getDefinitionType

Returns the definition type for the AmReceiver in the AmSubscriber.

```
int  getDefinitionType();
```

The following values can be returned:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

getEncoding

Returns the value used to encode numeric data types for the AmSender in the AmSubscriber. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the subscriber must convert the encoding of the message before it is sent.

```
int  getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmSubscriber.

```
AmString  getName();
```

getQueueName

Returns the queue name used by the AmSubscriber to receive messages. This is used to determine the queue name of a permanent dynamic AmReceiver in the AmSubscriber, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **setQueueName**.)

```
AmString  getQueueName();
```

open

Opens an AmSubscriber. **open** is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy &policy);
```

policy	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

receive

Receives a message, normally a publication, using the AmSubscriber. The message data, topic and other elements can be accessed using the message interface methods (see page 206).

receive is overloaded: the `pubMessage` parameter is required, but the others are optional.

```
void  receive(
  AmMessage   &pubMessage,
  AmMessage   &selectionMessage,
  AmPolicy    &policy);
```

pubMessage	The message object containing the data that has been published. The message object is reset implicitly before the receive takes place.
selectionMessage	A message object containing the correlation ID used to selectively receive a message from the AmSubscriber. If omitted, the first available message is received. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be be reset (see **AmMessage.setCorrelationId** on page 212) before invoking the receive method.
policy	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

setQueueName

Sets the queue name in the AmReceiver of the AmSubscriber, when this encapsulates a model queue. This is used to specify the queue name of a recreated permanent dynamic AmReceiver, in order to receive messages in a session subsequent to the one in which it was created. (See also **getQueueName**.)

```
void  setQueueName(char * queueName);
```

queueName	The queue name to be set.

subscribe

Sends a subscribe message to a publish/subscribe broker using the AmSubscriber, to register a subscription. The topic and other elements can be specified using the message interface methods (see page 206) before sending the message.

Publications matching the subscription are sent to the AmReceiver associated with the AmSubscriber. By default, this has the same name as the AmSubscriber, with the addition of the suffix '.RECEIVER'.

subscribe is overloaded: the `subMessage` parameter is required, but the others are optional.

```
void  subscribe(
  AmMessage   &subMessage,
  AmReceiver  &responseService,
  AmPolicy    &policy);
```

subMessage	The message object that contains the topic subscription data.
responseService	The AmReceiver to which the response to this subscribe request should be sent. Omit it if no response is required. This is not the AmReceiver to which publications will be sent by the broker; they are sent to the AmReceiver associated with the AmSubscriber (see above).
policy	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

unsubscribe

Sends an unsubscribe message to a publish/subscribe broker using the AmSubscriber, to deregister a subscription. The topic and other elements can be specified using the message interface methods (see page 206) before sending the message.

unsubscribe is overloaded: the `unsubMessage` parameter is required, but the others are optional.

```
void  unsubscribe(
  AmMessage   &unsubMessage,
  AmReceiver  &responseService,
  AmPolicy    &policy);
```

unsubMessage	The message object that contains the topics to which the unsubscribe request applies.
responseService	The AmReceiver to which the response to this unsubscribe request should be sent. Omit it if no response is required.
policy	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmPolicy

An **AmPolicy** object encapsulates details of how the AMI processes the message (for instance, the priority and persistence of the message, how errors are handled, and whether transactional processing is used).

clearErrorCodes

Clears the error codes in the AmPolicy.

```
void  clearErrorCodes();
```

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(AMBOOL warningsOn);
```

`warningsOn`	If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmPolicy object.

```
AmString  getName();
```

getWaitTime

Returns the wait time (in ms) set for this AmPolicy.

```
int  getWaitTime();
```

setWaitTime

Sets the wait time for any **receive** using this AmPolicy.

```
void  setWaitTime(int waitTime);
```

`waitTime`	The wait time (in ms) to be set in the AmPolicy.

AmBytes

An **AmBytes** object encapsulates an array of bytes. It allows the AMI to pass bytes across the interface and enables manipulation of these bytes.

cmp

Methods used to compare AmBytes objects. These methods return 0 if the data is the same, and 1 otherwise.

```
AMLONG cmp(const AmBytes &amBytes);
AMLONG cmp(const char * stringData);
AMLONG cmp(const char * charData, AMLONG length);
```

amBytes	A reference to the AmBytes object being compared.
stringData	A char pointer to the NULL terminated string being compared.
charData	A char pointer to the bytes being compared.
length	The length, in bytes, of the data to be compared. If this length is not the same as the length of the AmBytes object, the comparison fails.

constructors

Constructors for an AmBytes object.

```
AmBytes();
AmBytes(const AmBytes &amBytes);
AmBytes(const AMBYTE byte);
AmBytes(const AMLONG long);
AmBytes(const char * charData);
AmBytes(const AmString &amString);
AmBytes(const AMSTR stringData);
AmBytes(const AMBYTE *character, const AMLONG length);
```

amBytes	A reference to an AmBytes object used to create the new AmBytes object.
byte	A single byte used to create the new AmBytes object.
long	An AMLONG used to create the new AmBytes object.
charData	A char pointer to a NULL terminated string used to create the new AmBytes object.
stringData	A NULL terminated string used to create the new AmBytes object.
character	The character to populate the new AmBytes object with.
length	The length, in bytes, of the new AmBytes object.

cpy

Methods used to copy from an AmBytes object. Any existing data in the AmBytes object is discarded.

```
AmBytes &cpy();
AmBytes &cpy(const AMSTR stringData);
AmBytes &cpy(const AMBYTE *byteData, const AMLONG length);
AmBytes &cpy(const AMBYTE byte);
AmBytes &cpy(const AMLONG long);
AmBytes &cpy(const AmBytes &amBytes);
```

stringData	A NULL terminated string being copied.
byteData	A pointer to the bytes being copied.
length	The length, in bytes, of the data to be copied.
byte	The single byte being copied.
long	An AMLONG being copied.
amBytes	A reference to the AmBytes object being copied.

dataPtr

Method to reference the byte data contained within an AmBytes object.

```
const AMBYTE * dataPtr() const;
```

destructor

Destructor for an AmBytes object.

```
~AmBytes();
```

length

Returns the length of an AmBytes object.

```
AMLONG length();
```

operators

Operators for an AmBytes object.

```
AmBytes &operator = (const AmBytes &);
AMBOOL operator == (const AmBytes &) const;
AMBOOL operator != (const AmBytes &) const;
```

pad

Method used to pad AmBytes objects with a specified byte value.

```
AmBytes &pad(const AMLONG length, const AMBYTE byte);
```

length	The required length of the AmBytes after the padding.
byte	The byte value used to pad the AmBytes object.

AmElement

An **AmElement** object encapsulates a name/value pair which can be added to an AmMessage object.

AmElement

Constructor for an AmElement object.

```
AmElement(char * name, char * value);
```

`name`	The name of the element.
`value`	The value of the element.

getName

Returns the name of the AmElement.

```
AmString  getName();
```

getValue

Returns the value of the AmElement.

```
AmString  getValue();
```

getVersion

Returns the version of the AmElement (the default value is AMELEM_VERSION_1).

```
int  getVersion();
```

setVersion

Sets the version of the AmElement.

```
void  setVersion(int version);
```

`version`	The version of the AmElement that is set. It can take the value AMELEM_VERSION_1 or AMELEM_CURRENT_VERSION.

toString

Returns a AmString representation of the AmElement.

```
AmString  toString();
```

AmObject

AmObject is a virtual class. The following classes inherit from the AmObject class:

AmSession
AmMessage
AmSender
AmDistributionList
AmReceiver
AmPublisher
AmSubscriber
AmPolicy

This allows application programmers to use generic error handling routines.

clearErrorCodes

Clears the error codes in the AmObject.

```
void  clearErrorCodes();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmObject.

```
AmString  getName();
```

AmStatus

An **AmStatus** object encapsulates the error status of other AmObjects.

AmStatus

Constructor for an AmStatus object.

```
AmStatus();
```

getCompletionCode

Returns the completion code from the AmStatus object.

```
int  getCompletionCode();
```

getReasonCode

Returns the reason code from the AmStatus object.

```
int  getReasonCode();
```

getReasonCode2

Returns the secondary reason code from the AmStatus object. (This code is specific to the underlying transport used by the AMI). For WebSphere MQ, the secondary reason code is a WebSphere MQ reason code of type MQRC_xxx.

```
int  getReasonCode2();
```

toString

Returns an AmString representation of the internal state of the AmStatus object.

```
AmString  toString();
```

AmString

An **AmString** object encapsulates a string or array of characters. It allows the AMI to pass strings across the interface and enables manipulation of these strings.

cat

Methods used to concatenate.

```
AmString &cat(const AmString &amString);
AmString &cat(const AMSTR stringData);
```

amString	A reference to the AmString object being concatenated.
stringData	The NULL terminated string being concatenated into the AmString object.

cmp

Methods to compare AmStrings with AmStrings and data of type AMSTR. A return value of 0 indicates that the two strings match exactly.

```
AMLONG cmp(const AmString &amString) const;
AMLONG cmp(const AMSTR stringData) const;
```

amString	A reference to the AmString object being compared.
stringData	The NULL terminated string being compared.

constructors

Constructors for an AmString object.

```
AmString();
AmString(const AmString &amString);
AmString(const AMSTR stringData);
```

amString	A reference to an AmString object used to create the new AmString.
stringData	A NULL terminated string, from which the AmString is constructed.

contains

Method to indicate whether a specified character is contained within the AmString.

```
AMBOOL contains(const AMBYTE character) const;
```

character	The character being used for the search.

cpy

Methods used to copy from an AmString. Any existing data in the AmString is discarded.

```
AmString &cpy(const AmString &amString);
AmString &cpy(const AMSTR stringData);
```

amString	A reference to an AmString object being copied.
stringData	The NULL terminated string being copied into the AmString.

destructor

Destructor for an AmString object.

```
~AmString();
```

operators

Operators for an AmString object.

```
AmString &operator = (const AmString &);
AmString &operator = (const AMSTR);
AMBOOL operator == (const AmString &) const;
AMBOOL operator != (const AmString &) const;
```

pad

Method used to pad AmStrings with a specified character.

```
AmString &pad(const AMLONG length, const AMBYTE character);
```

`length`	The required length of the AmString after the padding.
`charString`	The character used to pad the AmString.

split

Method used to split AmStrings at the first occurrence of a specified character.

```
AmString &split(AmString &newString, const AMBYTE splitCharacter);
```

`newString`	A reference to an AmString object to contain the latter half of the split string.
`splitCharacter`	The first character at which the split will occur.

strip

Method used to strip leading and trailing blanks from AmStrings.

```
AmString &strip();
```

length

Returns the length of an AmString.

```
AMLONG length();
```

text

Method to reference the string contained within an AmString.

```
AMSTR text() const;
```

truncate

Method used to truncate AmStrings.

```
AmString &truncate(const AMLONG length);
```

`length`	The length to which the AmString is to be truncated.

AmException

AmException is the base Exception class; all other Exceptions inherit from this class.

getClassName

Returns the type of object throwing the Exception.

```
AmString  getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int  getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
AmString  getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int  getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject  getSource();
```

toString

Returns an AmString representation of the Exception.

```
AmString  toString();
```

AmErrorException

An Exception of type **AmErrorException** is raised when an object experiences an error with a severity level of FAILED (CompletionCode = AMCC_FAILED).

getClassName

Returns the type of object throwing the Exception.

```
AmString  getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int  getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
AmString  getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int  getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject  getSource();
```

toString

Returns an AmString representation of the Exception.

```
AmString  toString();
```

AmWarningException

An Exception of type **AmWarningException** is raised when an object experiences an error with a severity level of WARNING (CompletionCode = AMCC_WARNING).

getClassName

Returns the type of object throwing the Exception.

```
AmString  getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int  getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
AmString  getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int  getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject  getSource();
```

toString

Returns an AmString representation of the Exception.

```
AmString  toString();
```

Part 4. The COBOL interface

Chapter 10. Using the Application Messaging Interface in COBOL

The Application Messaging Interface (AMI) in the COBOL programming language has two interfaces:

1. A high-level procedural interface that provides the function needed by the majority of users.
2. A lower-level, object-style interface, that provides additional function for experienced WebSphere MQ users.

This chapter describes the following:

- "Structure of the AMI"
- "Writing applications in COBOL" on page 248
- "Building COBOL applications" on page 260

Structure of the AMI

Although the high-level interface is procedural in style, the underlying structure of the AMI is object based. (The term *object* is used here in the object-oriented programming sense, not in the sense of WebSphere MQ 'objects' such as channels and queues.) The objects that are made available to the application are:

Session	Contains the AMI session.
Message	Contains the message data, message ID, correlation ID, and options that are used when sending or receiving a message (most of which come from the policy definition).
Sender	This is a service that represents a destination (such as a WebSphere MQ queue) to which messages are sent.
Receiver	This is a service that represents a source from which messages are received.
Distribution list	Contains a list of sender services to provide a list of destinations.
Publisher	Contains a sender service where the destination is a publish/subscribe broker.
Subscriber	Contains a sender service (to send subscribe and unsubscribe messages to a publish/subscribe broker) and a receiver service (to receive publications from the broker).
Policy	Defines how the message should be handled, including items such as priority, persistence, and whether it is included in a unit of work.

When using the high-level functions the objects are created automatically and (where applicable) populated with values from the repository. In some cases it might be necessary to inspect these properties after a message has been sent (for instance, the *MessageID*), or to change the value of one or more properties before sending the message (for instance, the *Format*). To satisfy these requirements, the AMI for COBOL has a lower-level object style interface in addition to the high-level procedural interface. This provides access to the objects listed above, with methods to *set* and *get* their properties. You can mix high-level and object-level functions in the same application.

All the objects have both a *handle* and a *name*. The names are used to access objects from the high-level interface. The handles are used to access them from the object interface. Multiple objects of the same type can be created with the same name, but are usable only from the object interface.

The high-level interface is described in Chapter 11, "The COBOL high-level interface" on page 263. An overview of the object interface is given in Chapter 12, "COBOL object interface overview" on page 289, with reference information in Chapter 13, "COBOL object interface reference" on page 303.

Using the repository

You can run AMI applications with or without a repository. If you don't have a repository, you can use a system default object (see below), or create your own by specifying its name on a high-level function call. It will be created using the appropriate system provided definition (see "System provided definitions" on page 492).

If you have a repository, and you specify the name of an object on a function call that matches a name in the repository, the object will be created using the repository definition. (If no matching name is found in the repository, the system provided definition will be used.)

System default objects

Table 3. System default objects

Default object	Constant or handle (if applicable)
SYSTEM.DEFAULT.POLICY	AMSD-POL AMSD-POL-HANDLE
SYSTEM.DEFAULT.SYNCPOINT.POLICY	AMSD-SYNC-POINT-POL AMSD-SYNC-POINT-POL-HANDLE
SYSTEM.DEFAULT.SENDER	AMSD-SND
SYSTEM.DEFAULT.RESPONSE.SENDER	AMSD-RSP-SND AMSD-RSP-SND-HANDLE
SYSTEM.DEFAULT.RECEIVER	AMSD-RCV AMSD-RCV-HANDLE
SYSTEM.DEFAULT.PUBLISHER	AMSD-PUB AMSD-PUB-SND
SYSTEM.DEFAULT.SUBSCRIBER	AMSD-SUB AMSD-SUB-SND
SYSTEM.DEFAULT.SEND.MESSAGE	AMSD-SND-MSG AMSD-SND-MSG-HANDLE
SYSTEM.DEFAULT.RECEIVE.MESSAGE	AMSD-RCV-MSG AMSD-RCV-MSG-HANDLE

A set of system default objects is created at session creation time. This removes the overhead of creating the objects from applications using these defaults. The system default objects are available for use from both the high-level and object interfaces in COBOL. They are created using the system provided definitions (see "System provided definitions" on page 492).

The default objects can be specified explicitly using AMI constants, or used to provide defaults if a parameter is omitted (by specifying it as a space or low value, for example).

Constants representing synonyms for handles are also provided for these objects, for use from the object interface (see Appendix B, “Constants and structures” on page 561). Note that the first parameter on a call must be a real handle; you cannot use a synonym in this case (that is why handles are not provided for all the default objects).

Writing applications in COBOL

This section gives a number of examples showing how to use the high-level interface of the AMI, with some extensions using the object interface. Equivalent operations to all high-level functions can be performed using combinations of object interface functions (see “High-level functions” on page 301).

Opening and closing a session

Before using the AMI, you must open a session. This can be done with the following high-level function (page 271):

Opening a session

```
CALL 'AMHINIT' USING SESSION-NAME, POLICY-NAME, HSESSION,
                     COMPCODE, REASON.
```

The `SESSION-NAME` is optional. `POLICY-NAME` is the name of the policy to be used during initialization of the AMI. If it consists of a space or low value, the SYSTEM.DEFAULT.POLICY object is used. Or you can specify the constant AMSD-POL to use the default policy.

The function returns `HSESSION`, a *session handle* that must be used by other calls in this session. Errors are returned using a completion code and reason code.

To close a session, you can use this high-level function (page 286):

Closing a session

```
CALL 'AMHTERM' USING HSESSION, POLICY-NAME, COMPCODE, REASON.
```

This closes and deletes all objects that were created in the session.

Sending messages

You can send a datagram (send and forget) message using the high-level AMHSNMS function (page 282). In the simplest case, all you need to specify is the session handle returned by AMHINIT, the message data, and the message length. Other parameters can be specified using the constants that represent the default message, sender service, and policy objects.

Sending a message using all the defaults

```
CALL 'AMHSNMS' USING HSESSION, AMSD-SND, AMSD-POL, DATALEN, DATA,
                     AMSD-SND-MSG, COMPCODE, REASON.
```

If you want to send the message using a different sender service, specify its name (such as `SENDER-NAME`) as follows:

Sending a message using a specified sender service

```
CALL 'AMHSNMS' USING HSESSION, SENDER-NAME, AMSD-POL, DATALEN, DATA,
                     AMSD-SND-MSG, COMPCODE, REASON.
```

If you are not using the default policy, you can specify a policy name:

Sending a message using a specified policy

```
CALL 'AMHSNMS' USING HSESSION, AMSD-SND, POLICY-NAME, DATALEN, DATA,
                     AMSD-SND-MSG, COMPCODE, REASON.
```

The policy controls the behavior of the send function. For example, the policy can specify:
- The priority, persistence and expiry of the message
- If the send is part of a unit of work
- If the sender service should be implicitly opened and left open

To send a message to a distribution list, specify its name (such as `DISTLIST-NAME`) as the sender service:

Sending a message to a distribution list

```
CALL 'AMHSNMS' USING HSESSION, DISTLIST-NAME, AMSD-POL, DATALEN, DATA,
                     AMSD-SND-MSG, COMPCODE, REASON.
```

Using the message object

Using the object interface gives you more functions when sending a message. For example, you can *get* or *set* individual attributes in the message object. To get an attribute after the message has been sent, you can specify a name for the message object that is being sent:

Specifying a message object

```
CALL 'AMHSNMS' USING HSESSION, AMSD-SND, AMSD-POL, DATALEN, DATA,
                     SEND-MSG, COMPCODE, REASON.
```

The AMI creates a message object of the name specified (`SEND-MSG`), if one doesn't already exist. (In this example the defaults for the sender name and policy name are used.) You can then use object interface functions to get the required attributes, such as the *`MessageID`*, from the message object:

Getting an attribute from a message object

```
CALL 'AMSEGHMS' USING HSESSION, SEND-MSG, HMSG, COMPCODE, REASON.

CALL 'AMMSGTMI' USING HMSG, BUFFLEN, MSGIDLEN, MSGID, COMPCODE, REASON.
```

The first call is needed to get the handle to the message object (`HMSG`). The second call returns the message ID length, and the message ID itself (in a buffer of length `BUFFLEN`).

To set an attribute such as the *Format* before the message is sent, you must first create a message object and set the format in that object:

Setting an attribute in a message object

```
CALL 'AMSECRMS' USING HSESSION, SEND-MSG, HMSG, COMPCODE, REASON.

CALL 'AMMSSTFO' USING HMSG, FORMATLEN, FORMAT, COMPCODE, REASON.
```

Then you can send the message as before, making sure to specify the same message object name (`SEND-MSG`) in the AMHSNMS call.

Look at “Message interface functions” on page 292 to find out what other attributes of the message object you can get and set.

After a message object has been used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see AMMSRS on page 331) and rebuild it each time.

Instead of sending the message data using the data buffer, it can be added to the message object. However, this is not recommended for large messages because of the overhead of copying the data into the message object before it is sent (and also extracting the data from the message object when it is received).

Sample programs

For more details, refer to the AMTVHSND and AMTVOSND sample programs (see “Sample programs for z/OS” on page 483).

Receiving messages

Use the AMHRCMS high-level function (page 275) to receive a message to which no response is to be sent (such as a datagram). In the simplest case, all you need to specify are the session handle and a buffer for the message data. Other parameters can be specified using the constants that represent the default message, receiver service, and policy objects.

Receiving a message using all the defaults

```
CALL 'AMHRCMS' USING HSESSION, AMSD-RCV, AMSD-POL, AMSD-SND-MSG,
                     BUFFLEN,  DATALEN, DATA, AMSD-RCV-MSG,
                     COMPCODE, REASON.
```

If you want to receive the message using a different receiver service, specify its name (such as `RECEIVER-NAME`) as follows:

Receiving a message using a specified receiver service

```
CALL 'AMHRCMS' USING HSESSION, RECEIVER-NAME, AMSD-POL, AMSD-SND-MSG,
                     BUFFLEN,  DATALEN, DATA, AMSD-RCV-MSG,
                     COMPCODE, REASON.
```

If you are not using the default policy, you can specify a policy name:

Receiving a message using a specified policy

```
CALL 'AMHRCMS' USING HSESSION, AMSD-RCV, POLICY-NAME, AMSD-SND-MSG,
                     BUFFLEN,  DATALEN, DATA, AMSD-RCV-MSG,
                     COMPCODE, REASON.
```

The policy can specify, for example:
- The wait interval
- If the message is part of a unit of work
- If the message should be code page converted
- If all the members of a group must be there before any members can be read

Using the message object

To get the attributes of a message after receiving it, you can specify your own message object name, or use the system default SYSTEM.DEFAULT.RECEIVE.MESSAGE (constant: AMSD-RCV-MSG). If a message object of that name does not exist it will be created. You can access the attributes (such as the `Encoding`) using the object interface functions:

Getting an attribute from a message object

```
CALL 'AMHRCMS' USING HSESSION, AMSD-RCV, AMSD-POL, AMSD-SND-MSG,
                     BUFFLEN, DATALEN, DATA, RECEIVE-MSG,
                     COMPCODE, REASON.

CALL 'AMSEGHMS' USING HSESSION, RECEIVE-MSG, HMSG, COMPCODE, REASON.

CALL 'AMMSGTEN' USING HMSG, ENCODING, COMPCODE, REASON.
```

If a specific message is to be selectively received using its correlation identifier, a message object must first be created and its `CorrelId` property set to the required value (using the object interface). This message object is passed as the *selection message* on the AMHRCMS call:

Using a selection message object

```
CALL 'AMSECRMS' USING HSESSION, SELECTION-MSG, HMSG, COMPCODE, REASON.

CALL 'AMMSSTCI' USING HMSG, CORRELIDLEN, CORRELID, COMPCODE, REASON.

CALL 'AMHRCMS' USING HSESSION, AMSD-RCV, AMSD-POL, SELECTION-MSG,
                     BUFFLEN, DATALEN, DATA, AMSD-RCV-MSG,
                     COMPCODE, REASON.
```

Sample programs

For more details, refer to the AMTVHRCV and AMTVORCV sample programs (see “Sample programs for z/OS” on page 483).

Request/response messaging

In the *request/response* style of messaging, a requester (or client) application sends a request message and expects to receive a message in response. The responder (or server) application receives the request message and produces the response message (or messages) which it returns to the requester application. The responder application uses information in the request message to determine how to send the response message to the requester.

In the following examples 'CLIENT' refers to the requesting application, and 'SERVER' refers to the responding application.

Request

Use the AMHSNRQ high-level function (page 283) to send a request message. This is similar to AMHSNMS, but it includes the name of the service to which the response message is to be sent. In this example the sender service (`CLIENT-SENDER`) is specified in addition to the receiver service (`CLIENT-RECEIVER`). A send message name (`CLIENT-SND-MSG`) is specified as well.

Sending a request message

```
CALL 'AMHSNRQ' USING HSESSION, CLIENT-SENDER, AMSD-POL, CLIENT-RECEIVER,
                     DATALEN, DATA, CLIENT-SND-MSG, COMPCODE, REASON.
```

The AMHRCRQ high-level function (page 279) is used by the responding (or server) application to receive a request message. It is similar to AMHRCMS, but it includes the name of the sender service that will be used for sending the response message. When the message is received, the sender service is updated with the information needed for sending the response to the required destination.

Receiving a request message

```
CALL 'AMHRCRQ' USING HSESSION, SERVER-RECEIVER, AMSD-POL, BUFFLEN,
                     DATALEN, DATA, SERVER-RCV-MSG, SERVER-SENDER,
                     COMPCODE, REASON.
```

A policy name can be specified as well, as described in "Receiving messages" on page 250.

A receiver message name (`SERVER-RCV-MSG`) is specified so that the response message can refer to it. Note that, unlike AMHRCMS, this function does not have a selection message.

Response

After the requested actions have been performed, the responding application sends the response message (or messages) with the AMHSNRS function (page 284):

Sending a response message

```
CALL 'AMHSNRS' USING HSESSION, SERVER-SENDER, AMSD-POL, SERVER-RCV-MSG,
                     DATALEN, DATA, AMSD-SND-MSG, COMPCODE, REASON.
```

The sender service for the response message (`SERVER-SENDER`) and the receiver message name (`SERVER-RCV-MSG`) are the same as those used with AMHRCRQ (receive request). This causes the *CorrelId* and *MessageId* to be set in the response message, as requested by the flags in the request message.

Finally, the requester (or client) application uses the AMHRCMS function to receive the response message as described in "Receiving messages" on page 250. You might need to receive a specific response message (for example if three request messages have been sent, and you want to receive the response to the first request message first). In this case the sender message name from the AMHSNRQ function (`CLIENT-SND-MSG`) should be used as the selection message name in AMHRCMS.

Sample programs

For more details, refer to the AMTVHCLT, AMTVOCLT, AMTVHSVR, and AMTSOSVR sample programs (see "Sample programs for z/OS" on page 483).

File transfer

You can perform file transfers using the AMHSNFL and AMHRCFL high-level functions, and the AMSNSNFL, AMDLSNFL and AMRCRCFL object-level functions.

Sending a file using the high-level AMHSNFL function

```
CALL 'AMHSNFL' USING HSESSION, SENDER-NAME, POLICYNAME, OPTIONS,
                     FILENAME-LENGTH, FILENAME, SNDMSG-NAME.
```

Receiving a file using the high-level AMHRCFL function

```
CALL 'AMHRCFL' USING HSESSION, RECEIVER-NAME, POLICY-NAME, OPTIONS,
                     SELMSG-NAME, FILENAME-LENGTH, SNDMSG-NAME.
```

For a complete description of file transfer, refer to "File transfer" on page 21.

Publish/subscribe messaging

With *publish/subscribe* messaging, *publisher* applications publish messages to *subscriber* applications using a *broker*. The messages published contain application data and one or more *topic* strings that describe the data. Subscribing applications register subscriptions informing the broker which topics they are interested in. When the broker receives a published message, it forwards the message to all subscribing applications for which a topic in the message matches a topic in the subscription.

Subscribing applications can exploit content-based publish/subscribe by passing a filter on subscribe and unsubscribe calls (see "Using WebSphere MQ Integrator Version 2" on page 477).

For more information, refer to the *WebSphere MQ Publish/Subscribe User's Guide*.

Publish

Use the AMHPB high-level function (page 272) to publish a message. You need to specify the name of the publisher for the publish/subscribe broker (or use the default by specifying AMSD-PUB).

The topic relating to this publication and the publication data must also be specified:

Publishing a message

```
CALL 'AMHPB' USING HSESSION, PUBLISHER-NAME, AMSD-POL, RECEIVER-NAME,
                   TOPICLEN, TOPIC, DATALEN, DATA, PUBLISH-MSG,
                   COMPCODE, REASON.
```

The `RECEIVER-NAME` identifies the receiver service to which the broker will send a response message. You can also specify a policy name to change the behavior of the function (as with the AMHSNxx functions).

You can specify the publication message name `PUBLISH-MSG` and set or get attributes of the message object (using the object interface functions). This might include adding another topic (using AMMSADTO) before invoking AMHPB, if there are multiple topics associated with this publication.

Instead of sending the publication data using the data buffer, it can be added to the message object. Unlike the AMHSNxx functions, this gives no difference in performance with large messages. This is because, whichever method is used, the MQRFH header has to be added to the publication data before sending it (similarly the header has to be removed when the publication is received).

Subscribe

The AMHSB high-level function (page 285) is used to subscribe to a publish/subscribe broker specified by the name of a subscriber service. The receiver to which publications will be sent is included within the definition of the subscriber. The name of a receiver service to which the broker can send a response message (`RECEIVER-NAME`) is also specified.

Subscribing to a broker

```
CALL 'AMHSB' USING HSESSION, SUBSCRIBER-NAME, AMSD-POL, RECEIVER-NAME,
                   TOPICLEN, TOPIC, 0, 0, SUBSCRIBE-MSG,
                   COMPCODE, REASON.
```

A subscription for a single topic can be passed by the `TOPIC` parameter. You can subscribe to multiple topics by using the object interface AMMSADTO function to add topics to the `SUBSCRIBE-MSG` message object, before invoking AMHSB.

If the policy specifies that the *`CorrelId`* is to be used as part of the identity for the subscribing application, it can be added to the subscription message object with the object interface AMMSSTCI function, before invoking AMHSB.

To remove a subscription, use the AMHUN high-level function (page 287). To remove all subscriptions, you can specify a policy that has the ‘Deregister All Topics’ subscriber attribute.

To receive a publication from a broker, use the AMHRCPB function (page 277). For example:

Receiving a publication

```
CALL 'AMHRCPB' USING HSESSION, SUBSCRIBER-NAME, AMSD-POL, SELECTION-MSG,
                     TOPICBUFFLEN, BUFFLEN, TOPICCOUNT, TOPICLEN,
                     FIRSTTOPIC, DATALEN, DATA, RECEIVE-MSG,
                     COMPCODE, REASON.
```

You need to specify the name of the subscriber service used for the original subscription. You can also specify a policy name and a selection message name, as described in “Receiving messages” on page 250.

If there are multiple topics associated with the publication, only the first one is returned by this function. So, if `TOPICCOUNT` indicates that there are more topics, you have to access them from the `RECEIVE-MSG` message object, using the object-level AMSEGHMS (get message handle) and AMMSGTTO (get topic) functions.

Sample programs

For more details, refer to the AMTVHPUB, AMTSOPUB, AMTVHSUB, and AMTSOSUB sample programs (see “Sample programs for z/OS” on page 483).

Using name/value elements

Publish/subscribe brokers (such as WebSphere MQ Publish/Subscribe) respond to messages that contain name/value pairs to define the commands and options to be used. The AMHPB, AMHSB, AMHUN, and AMHRCPB high-level functions provide these name/value pairs implicitly.

For less commonly used commands and options, the name/value pairs can be added to a message using an AMELEM structure. The AMTELEMV and AMTELEML copybooks define the AMELEM structure, with and without default values respectively. Here is the AMTELEMV copybook:

```
**   AMELEM structure
  10 AMELEM.
**    Structure identifier
   15 AMELEM-STRUCID                PIC X(8) VALUE 'COEL    '.
**    Structure version number
   15 AMELEM-VERSION                PIC S9(9) BINARY VALUE 1.
**    Reserved, must be zero
   15 AMELEM-GROUP-BUFF-LEN         PIC S9(9) BINARY VALUE 0.
**    Reserved, must be zero
   15 AMELEM-GROUP-LEN              PIC S9(9) BINARY VALUE 0.
**    Reserved, must be zero
   15 AMELEM-GROUP-OFFSET           PIC S9(9) BINARY VALUE 0.
**    Name buffer length
   15 AMELEM-NAME-BUFF-LEN          PIC S9(9) BINARY VALUE 0.
**    Name length in bytes
   15 AMELEM-NAME-LEN               PIC S9(9) BINARY VALUE 0.
**    Name
   15 AMELEM-NAME-OFFSET            PIC S9(9) BINARY VALUE 0.
**    Value buffer length
   15 AMELEM-VALUE-BUFF-LEN         PIC S9(9) BINARY VALUE 0.
**    Value length in bytes
   15 AMELEM-VALUE-LEN              PIC S9(9) BINARY VALUE 0.
**    Value
   15 AMELEM-VALUE-OFFSET           PIC S9(9) BINARY VALUE 0.
**    Reserved, must be zero
```

```
    15 AMELEM-TYPE-BUFF-LEN          PIC S9(9) BINARY VALUE 0.
 **    Reserved, must be zero
    15 AMELEM-TYPE-LEN               PIC S9(9) BINARY VALUE 0.
 **    Reserved, must be zero
    15 AMELEM-TYPE-OFFSET            PIC S9(9) BINARY VALUE 0.
```

The offset fields in the AMELEM structure allow you to give the location of the name and value buffers relative to the start of the AMELEM structure. The offsets can be positive or negative.

Following are short descriptions of the fields and an example of how to use the AMELEM structure.

AMELEM-STRUCID
The AMELEM structure identifier (input).

AMELEM-VERSION
The version number of the AMELEM structure (input). Its value must be one.

AMELEM-GROUP-BUFF-LEN
Reserved, must be zero.

AMELEM-GROUP-LEN
Reserved, must be zero.

AMELEM-GROUP-OFFSET
Reserved, must be zero.

AMELEM-NAME-BUFF-LEN
The length of the name buffer (input). If this field is set to zero, the AMI returns the name length value (in AMELEM-NAME-LEN) but not the name value (in AMELEM-NAME-OFFSET). This is not an error.

AMELEM-NAME-LEN
The length of the name in bytes (input or output).

AMELEM-NAME-OFFSET
The name buffer (input or output).

AMELEM-VALUE-BUFF-LEN
The length of the value buffer (input).

AMELEM-VALUE-LEN
The value length in bytes (input or output).

AMELEM-VALUE-OFFSET
The value buffer (input or output).

AMELEM-TYPE-BUFF-LEN
Reserved, must be zero.

AMELEM-TYPE-LEN
Reserved, must be zero.

AMELEM-TYPE-OFFSET
Reserved, must be zero.

Example

As an example, to send a message containing a 'Request Update' command, define the command data and the AMELEM structure as follows:

```
01  OPTIONS                     PIC S9(9) BINARY VALUE ZERO.
01  AMELEM-DATA.
    10  COMMAND-NAME            PIC X(16) VALUE 'MQPSCommand'.
    10  COMMAND-VALUE           PIC X(16) VALUE 'ReqUpdate'
    COPY AMTELEMV.
```

Set the length and offset values as follows:

```
MOVE 11  TO AMELEM-NAME-LEN.
MOVE -48 TO AMELEM-NAME-OFFSET.
MOVE 9   TO AMELEM-VALUE-LEN.
MOVE -32 TO AMELEM-VALUE-OFFSET.
```

Having set the values, create a message object (`SEND-MSG`) and add the element to it:

Using name/value elements

```
CALL 'AMSECRMS' USING HSESSION, SEND-MSG, HMSG, COMPCODE, REASON.

CALL 'AMMSADEL' USING HMSG, AMELEM, OPTIONS, COMPCODE, REASON.
```

You must then send the message, using AMHSNMS, to the sender service specified for the publish/subscribe broker.

If you need to use streams with WebSphere MQ Publish/Subscribe, you must add the appropriate stream name/value element explicitly to the message object.

The message element functions can, in fact, be used to add any element to a message before issuing a publish/subscribe request. Such elements (including topics, which are specialized elements) supplement or override those added implicitly by the request, as appropriate to the individual element type.

The use of name/value elements is not restricted to publish/subscribe applications. They can be used in other applications as well.

Error handling

Each AMI COBOL function returns a completion code reflecting the success or failure (OK, warning, or error) of the request. Information indicating the reason for a warning or error is returned in a reason code.

The 'get last error' functions (such as AMSEGTLE) always reflect the last most severe error detected by an object. These functions can be used to return the completion and reason codes associated with this error. Once the error has been handled, call the 'clear error codes' functions (for instance, AMMSCLEC) to clear the error information.

All COBOL high-level functions record last error information in the session object. This information can be accessed using the session's 'get last error' call, AMSEGTLE (you need the session handle returned by AMHINIT as the first parameter of this call).

Transaction support

Messages sent and received by the AMI can, optionally, be part of a transactional unit of work. A message is included in a unit of work based on the setting of the syncpoint attribute specified in the policy used on the call. The scope of the unit of work is the session handle and only one unit of work may be active at any time.

The API calls used to control the transaction depends on the type of transaction is being used.

- WebSphere MQ messages are the only resource

  This is supported under z/OS batch. A transaction is started by the first message sent or received under syncpoint control, as specified in the policy specified for the send or receive. Multiple messages can be included in the same unit of work. The transaction is committed or backed out using an AMHCMIT or AMHBACK high-level interface call (or the AMSECM or AMSERB object-level calls).
- Using an external transaction coordinator

  The transaction is controlled using the API calls of an external transaction coordinator. Supported coordinators are CICS, IMS, and RRS. The AMI calls are not used but the syncpoint attribute must still be specified in the policy used on the call.

Sending group messages

The AMI allows a sequence of related messages to be included in, and sent as, a message group. Group context information is sent with each message to allow the message sequence to be preserved and made available to a receiving application. To include messages in a group, the group status information of the first and subsequent messages in the group must be set as follows:

```
AMGRP_FIRST_MSG_IN_GROUP for the first message
AMGRP_MIDDLE_MSG_IN_GROUP for all messages other than first and last
AMGRP_LAST_MSG_IN_GROUP for the last message
```

The message status is set using **AMMSSTGS**.

For a complete description of group messages, refer to “Sending group messages” on page 27.

Applications for z/OS subsystems

If you want to write AMI applications for the IMS, CICS, batch, and RRS-batch subsystems on z/OS, refer to “Applications for z/OS subsystems” on page 27.

Other considerations

You should consider the following when writing your applications:
- Multithreading
- Using WebSphere MQ with the AMI
- Field limits

Multithreading

Multithreading is not supported for COBOL applications running on z/OS.

Using WebSphere MQ with the AMI

You must not mix WebSphere MQ function calls with AMI function calls within the same process.

Field limits

When string and binary properties such as queue name, message format, and correlation ID are set, the maximum length values are determined by WebSphere MQ, the underlying message transport. See the rules for naming WebSphere MQ objects in the *WebSphere MQ Application Programming Guide*.

Building COBOL applications

The Application Messaging Interface for COBOL is available only on the z/OS operating system.

COBOL applications on z/OS

This section explains what you have to do to prepare and run your COBOL programs on the z/OS operating system. See *WebSphere MQ for z/OS Concepts and Planning Guide* for compilers supported by the AMI.

AMI Copybooks

The AMI provides COBOL copybooks to assist you with the writing of your applications. The copybook AMTV contains constants and return codes. Copybooks AMTELEML and AMTELEMV contain the definition of the AMELEM data structure that is used to pass name/value element information across the AMI. AMTELEML provides a data definition without initial values; AMTELEMV provides the same definition with initial values.

These copybooks are installed in the WebSphere MQ for z/OS library hlq.SCSQCOBC. Use the COPY statement to include them in your program. For example:

```
WORKING STORAGE SECTION.
01  AMI-CONSTANTS.
    COPY AMTV.
```

You are recommended to use the copybook AMTELEMV to define an AMELEM structure. This provides default initial values which ensures that the *`strucId`* and *`version`* fields have valid values. If the values passed for these fields are not valid, the AMI will reject them.

Preparing COBOL programs on z/OS

COBOL programs that use the AMI must be compiled and linked edited. Programs containing CICS commands must be processed by the CICS translator before compilation. To add AMI support, include the appropriate COBOL stub (interface module) in the link edit. The AMI provides a COBOL stub for each supported environment (batch, RRS batch, or CICS), as follows:

Batch	AMTBS10
RRS batch	AMTRS10
CICS	AMTCS10
IMS	AMTIS10

Note: Select the NODYNAM compiler option to enable the linkage editor to resolve references to the AMI stub.

Thus the link edit JCL should specify a ‘DD’ name for the WebSphere MQ for z/OS hlq.SCSQLOAD library and an INCLUDE statement for the stub. For example, to link edit an AMI batch application:

```
//LKED    EXEC PGM=HEWL....
....
//OBJLIB  DD DSN=thlqual.SCSQLOAD,DISP=SHR
//SYSIN   DD *
  ENTRY CEESTART
  INCLUDE OBJLIB(AMTBS10)
  NAME progname(R)
/*
```

Running COBOL programs on z/OS

The AMI needs access to the WebSphere MQ datasets SCSQLOAD and SCSQAUTH, as well as one of the language-specific datasets such as SCSQANLE. See the *WebSphere MQ Application Programming Guide* for details of the supported languages.

For CICS operation, the library hlq.SCSQLOAD and the Language Environment® SCEERUN library must be included in the DFHRPL concatenation. COBOL programs using the AMI must be defined to CICS with a language code of ‘Le370’.

For information about AMI tracing, see “Using trace (z/OS)” on page 529.

COBOL applications on z/OS

Chapter 11. The COBOL high-level interface

The COBOL high-level interface contains functions that cover the requirements of the majority of applications. If extra functionality is needed, COBOL object interface functions can be used in the same application as the COBOL high-level functions.

This chapter contains:
- “Overview of the COBOL high-level interface” on page 264
- “Reference information for the COBOL high-level interface” on page 265

Overview of the COBOL high-level interface

The high-level functions are listed below. Follow the page references to see the detailed descriptions of each function.

Initialize and terminate

Functions to create and open an AMI session, and to close and delete an AMI session.

AMHINIT (initialize)	page 271
AMHTERM (terminate)	page 286

Sending messages

Functions to send a datagram (send and forget) message, and to send request and response messages.

AMHSNMS (send message)	page 282
AMHSNRQ (send request)	page 283
AMHSNRS (send response)	page 284

Receiving messages

Functions to receive a message from AMHSNMS or AMHSNRS, to receive a request message from AMHSNRQ, and to browse a message.

AMHRCMS (receive message)	page 275
AMHRCRQ (receive request)	page 279
AMHBRMS (browse message)	page 268

File transfer

Functions to send message data from a file, and to receive message data sent by AMHSNFL into a file.

AMHSNFL (send file)	page 281
AMHRCFL (receive file)	page 273

Publish/subscribe

Functions to publish a message to a publish/subscribe broker, and to subscribe, unsubscribe, and receive publications.

AMHPB (publish)	page 272
AMHSB (subscribe)	page 285
AMHUN (unsubscribe)	page 287
AMHRCPB (receive publication)	page 277

Transaction support

Functions to begin, commit and back out a unit of work.

AMHBEGIN (begin)	page 267

AMHCMIT (commit)	page 270
AMHBACK (backout)	page 266

Reference information for the COBOL high-level interface

In the following sections the high-level interface functions are listed in alphabetical order. Note that all functions return a completion code (COMPCODE) and a reason code (REASON). The completion code can take one of the following values:

AMCC-OK	Function completed successfully
AMCC-WARNING	Function completed with a warning
AMCC-FAILED	An error occurred during processing

If the completion code returns warning or failed, the reason code identifies the reason for the error or warning (see Appendix A, “Reason codes and LDAP error codes” on page 537).

Object names can be up to AMLEN-MAX-NAME-LENGTH characters, and are terminated by a space or by a low value (a single byte zero). If a space or low value is not found, the name will be truncated at AMLEN-MAX-NAME-LENGTH.

If an object name is specified as a space or low value, the relevant system default name will be used.

Most functions require the session handle to be specified. If this handle is not valid, the results are unpredictable.

AMHBACK (backout)

Function to back out a unit of work.

```
  CALL 'AMHBACK' USING HSESSION, POLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  POLICY        PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHBEGIN (begin)

Function to begin a unit of work.

```
  CALL 'AMHBEGIN' USING HSESSION, POLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  POLICY        PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHBRMS (browse message)

Function to browse a message. See the *WebSphere MQ Application Programming Guide* for a full description of the browse options.

```
CALL 'AMHBRMS' USING HSESSION, RECEIVER, POLICY, OPTIONS,
                     BUFFLEN, DATALEN, DATA, RCVMSGNAME,
                     SENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  RECEIVER      PIC X(n).
01  POLICY        PIC X(n).
01  OPTIONS       PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  RCVMSGNAME    PIC X(n).
01  SENDER        PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION The session handle returned by AMHINIT (input).

RECEIVER The name of a receiver service (input). If specified as a space or low value, the system default receiver name (constant: AMSD-RCV) is used.

POLICY The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.

OPTIONS Options controlling the browse operation (input). Possible values are:

```
AMBRW-NEXT
AMBRW-FIRST
AMBRW-RECEIVE-CURRENT
AMBRW-DEFAULT          (AMBRW-NEXT)
```

`AMBRW-RECEIVE-CURRENT` is equivalent to AMRCRC (receive) for the message under the browse cursor.

BUFFLEN The length in bytes of a buffer in which the data is returned (input).

DATALEN The length of the message data, in bytes (output). Can be specified as -1 (input).

DATA The received message data (output).

RCVMSGNAME The name of the message object for the received message (input). Properties, and message data if not returned in the `DATA` parameter, can be extracted from the message object using the object interface (see “Message interface functions” on page 317). The message object is implicitly reset before the browse takes place. If specified as a space or low value, the system default receive message name (constant: AMSD-RCV-MSG) is used.

SENDER The name of a special type of sender service known as a *response sender*, to which the response message will be sent (input). This sender name must not have been defined in the repository before the start of the AMI session. It is only applicable if the message type is AMMT-REQUEST.

COMPCODE Completion code (output).

REASON	Reason code (output).

Usage notes

You can return data in the message object or in an application buffer.

To return the data in the message object (RCVMSGNAME), rather than the application message buffer, set `BUFFLEN` to zero and set both `DATA` and `DATALEN` as non_NULL (not −1).

To return data in an application message buffer:

- set `DATA` as the address of the buffer (that is, non_NULL, not −1)
- set `BUFFLEN` to the length of the buffer

If the value of `BUFFLEN` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a non_NULL value for `DATALEN` (that is, not −1).

To return only the data length:

- set `DATA` to NULL (−1)
- set `BUFFLEN` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

AMHCMIT (commit)

Function to commit a unit of work.

```
  CALL 'AMHCMIT' USING HSESSION, POLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  POLICY        PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHINIT (initialize)

Function to create and open an AMI session. It returns a session handle, which is valid until the session is terminated.

```
CALL 'AMHINIT' USING SESSNAME, POLICY, HSESSION, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  SESSNAME      PIC X(n).
01  POLICY        PIC X(n).
01  HSESSION      PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

SESSNAME	An optional name that can be used to identify the application (input).
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
HSESSION	The session handle (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHPB (publish)

Function to publish a message to a publish/subscribe broker.

```
CALL 'AMHPB' USING HSESSION, PUBLISHER, POLICY, RESPNAME,
                   TOPICLEN, TOPIC, DATALEN, DATA, MSGNAME,
                   COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  PUBLISHER     PIC X(n).
01  POLICY        PIC X(n).
01  RESPNAME      PIC X(n).
01  TOPICLEN      PIC S9(9) BINARY.
01  TOPIC         PIC X(n).
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  MSGNAME       PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
PUBLISHER	The name of a publisher service (input). If specified as a space or low value, the system default publisher name (constant: AMSD-PUB) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
RESPNAME	The name of the receiver service to which the response to this publish request will be sent (input). If specified as a space or low value, no response will be sent. This parameter is mandatory if the policy specifies implicit publisher registration (the default).
TOPICLEN	The length of the topic for this publication, in bytes (input).
TOPIC	The topic for this publication (input).
DATALEN	The length of the publication data in bytes (input). A value of zero indicates that any publication data has been added to the message object (`MSGNAME`) using the object interface (see “Message interface functions” on page 317).
DATA	The publication data, if `DATALEN` is non-zero (input).
MSGNAME	The name of a message object that contains the header for the publication message (input). If `DATALEN` is zero, the message object also holds any publication data. If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHRCFL (receive file)

Function to receive message data sent by AMHSNFL into a file.

```
CALL 'AMHRCFL' USING HSESSION, RECEIVERNAME, POLICYNAME,
                     OPTIONS, SELMSGNAME, DIRNAMELEN,
                     DIRNAME, FILENAMELEN, FILENAME,
                     RCVMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  RECEIVERNAME  PIC X(n).
01  POLICYNAME    PIC X(n).
01  OPTIONS       PIC S9(9) BINARY.
01  SELMSGNAME    PIC X(n).
01  DIRNAMELEN    PIC S9(9) BINARY.
01  DIRNAME       PIC X(n).
01  FILENAMELEN   PIC S9(9) BINARY.
01  FILENAME      PIC X(n).
01  RCVMSGNAME    PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
RECEIVERNAME	The name of a receiver service (input). If specified as a space or low value, the system default receiver name (constant: AMSD-RCV) is used.
POLICYNAME	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
OPTIONS	Reserved, must be specified as zero.
SELMSGNAME	Optional selection message object used to specify information (such as a `CorrelId`) needed to select the required message (input).
DIRNAMELEN	Reserved, must be specified as zero (input).
DIRNAME	Reserved.
FILENAMELEN	The length of the file name in bytes (input). .
FILENAME	The name of the file into which the transferred data is to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If blank then the AMI will use the name of the originating file (including any directory prefix) exactly as it was supplied on the send file call. Note that the original file name may not be appropriate for use by the receiver, either because a path name included in the file name is not applicable to the receiving system, or because the sending and receiving systems use different file naming conventions.
RCVMSGNAME	The name of the message object to be used to receive the file (output). This parameter is updated with the message properties (for example, the Message ID). If the message is not from a file, `rcvMsgName` receives the message data. If specified as a blank or low value, the system default receive message name (constant AMSD-RCV-MSG) is used. Property information and message data can be extracted from the message object using the object interface (see “Message interface functions” on page 317). The message object is reset implicitly before the receive takes place.

COBOL high-level interface

COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

If `FILENAME` is blank (indicating that the originating file name specified in the message is to be used), `FILENAMELEN` should be set to zero.

AMHRCMS (receive message)

Function to receive a message.

```
CALL 'AMHRCMS' USING HSESSION, RECEIVER, POLICY, SELMSGNAME,
                     BUFFLEN, DATALEN, DATA, RCVMSGNAME,
                     COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  RECEIVER      PIC X(n).
01  POLICY        PIC X(n).
01  SELMSGNAME    PIC X(n).
01  BUFFLEN       PIC S9(9) BINARY.
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  RCVMSGNAME    PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
RECEIVER	The name of a receiver service (input). If specified as a space or low value, the system default receiver name (constant: AMSD-RCV) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
SELMSGNAME	Optional selection message object used to specify information (such as a *CorrelId*) needed to select the required message (input).
BUFFLEN	The length in bytes of a buffer in which the data is returned (input).Can be specified as -1.
DATALEN	The length of the message data, in bytes (output). Can be specified as -1 (input).
DATA	The received message data (output).
RCVMSGNAME	The name of the message object for the received message (output). If specified as a space or low value, the system default receive message name (constant: AMSD-RCV-MSG) is used. Properties, and message data if not returned in the `DATA` parameter, can be extracted from the message object using the object interface (see “Message interface functions” on page 317). The message object is implicitly reset before the receive takes place.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

You can return data in the message object or in an application buffer.

To return the data in the message object (RCVMSGNAME), rather than the application message buffer, set `BUFFLEN` to zero and set both `DATA` and `DATALEN` as non_NULL (not –1).

To return data in an application message buffer:

- set `DATA` as the address of the buffer (that is, non_NULL, not –1)
- set `BUFFLEN` to the length of the buffer

If the value of `BUFFLEN` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a non_NULL value for `DATALEN` (that is, not −1).

To return only the data length without removing the message from the queue:
- set `DATA` to NULL (−1)
- set `BUFFLEN` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

To remove the message from the queue and discard it:
- set `DATA` or `DATALEN` to a non_NULL value (that is, not −1)
- set `BUFFLEN` to zero
- ensure that Accept Truncated Message in the policy receive attributes is selected

The message will be discarded with an AMRC_MSG_TRUNCATED warning.

If AMRC_RECEIVE_BUFF_LEN_ERR is returned, the message length value is returned in `DATALEN` (if it is non_NULL, that is, not −1), even though the completion code is MQCC_FAILED.

Note that if `DATA` is NULL (−1) and `BUFFLEN` is not zero, there is always an AMRC_RECEIVE_BUFF_LEN_ERR error.

AMHRCPB (receive publication)

Function to receive a publication from a publish/subscribe broker.

```
CALL 'AMHRCPB' USING HSESSION, SUBSCRIBER, POLICY, SELMSGNAME,
                     TOPICBUFFLEN, BUFFLEN, TOPICCOUNT, TOPICLEN,
                     FIRSTTOPIC, DATALEN, DATA, RCVMSGNAME,
                     COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  SUBSCRIBER    PIC X(n).
01  POLICY        PIC X(n).
01  SELMSGNAME    PIC X(n).
01  TOPICBUFFLEN  PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  TOPICCOUNT    PIC S9(9) BINARY.
01  TOPICLEN      PIC S9(9) BINARY.
01  FIRSTTOPIC    PIC X(n).
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  RCVMSGNAME    PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SUBSCRIBER	The name of a subscriber service (input). If specified as a space or low value, the system default subscriber name (constant: AMSD-SUB) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
SELMSGNAME	Optional selection message object used to specify information (such as a *CorrelId*) needed to select the required message (input).
TOPICBUFFLEN	The length in bytes of a buffer in which the topic is returned (input).
BUFFLEN	The length in bytes of a buffer in which the publication data is returned (input).
TOPICCOUNT	The number of topics in the message (output).
TOPICLEN	The length in bytes of the first topic (output).
FIRSTTOPIC	The first topic (output). Topics can be extracted from the message object (`RCVMSGNAME`) using the object interface (see "Message interface functions" on page 317).
DATALEN	The length in bytes of the publication data (output).
DATA	The publication data (output). Data can be extracted from the message object (`RCVMSGNAME`) using the object interface (see "Message interface functions" on page 317).
RCVMSGNAME	The name of a message object for the received message (input). If specified as a space or low value, the system default message name (constant: AMSD-RCV-MSG) is used. The publication message properties and data update this message object, in addition to being returned in the parameters above. The message object is implicitly reset before the receive takes place.
COMPCODE	Completion code (output).

`REASON`	Reason code (output).

Usage notes

We recommend that, when using `AMHRCPB`, you always have data conversion enabled in the specified policy. If data conversion is not enabled, `AMHRCPB` will fail if the local CCSID and/or encoding values differ from those on the platform from which the publication was sent.

If data conversion is enabled by the specified policy, and a selection message is specified, the conversion is performed using the target encoding and coded character set identifier (CCSID) values designated in the selection message. (The selection message is specified in the `SELMSGNAME` parameter).

If a selection message is not specified, the platform encoding and Queue Manager CCSID values are used as defaults for the conversion.

If a normal message that is not a publication message is received by the specified subscriber, `AMHRCPB` behaves the same as `AMHRCMS`.

AMHRCRQ (receive request)

Function to receive a request message.

```
CALL 'AMHRCRQ' USING HSESSION, RECEIVER, POLICY, BUFFLEN, DATALEN,
                     DATA, RCVMSGNAME, SENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  RECEIVER      PIC X(n).
01  POLICY        PIC X(n).
01  BUFFLEN       PIC S9(9) BINARY.
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  RCVMSGNAME    PIC X(n).
01  SENDER        PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
RECEIVER	The name of a receiver service (input). If specified as a space or low value, the system default receiver name (constant: AMSD-RCV) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
BUFFLEN	The length in bytes of a buffer in which the data is returned (input).
DATALEN	The length of the message data, in bytes (output). Can be specified as -1 (input).
DATA	The received message data (output).
RCVMSGNAME	The name of the message object for the received message (output). If specified as NULL, the system default receiver service (constant: AMSD-RCV-MSG) is used. Header information, and message data if not returned in the `DATA` parameter, can be extracted from the message object using the object interface (see “Message interface functions” on page 317). The message object is implicitly reset before the receive takes place.
SENDER	The name of a special type of sender service known as a *response sender*, to which the response message will be sent (output). This sender name must not be defined in the repository. If specified as a space or low value, the system default response sender service (constant: AMSD-RSP-SND) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

The following notes contain details about use of the `AMHRCRQ` function.

Data conversion

If data conversion is enabled by the specified policy, and a selection message is specified, the conversion is performed using the target encoding and coded character set identifier (CCSID) values designated in the selection message. (These target values are specified in the `SELMSGNAME` parameter).

If a selection message is not specified, the platform encoding and Queue Manager CCSID values are used as defaults for the conversion.

Use of the buffLen parameter

You can return data in the message object or in an application buffer.

To return the data in the message object (RCVMSGNAME), rather than the application message buffer, set `BUFFLEN` to zero and set both `DATA` and `DATALEN` as non_NULL (not −1).

To return data in an application message buffer:

- set `DATA` as the address of the buffer (that is, non_NULL, not −1)
- set `BUFFLEN` to the length of the buffer

If the value of `BUFFLEN` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a non_NULL value for `DATALEN` (that is, not −1).

To return only the data length without removing the message from the queue:

- set `DATA` to NULL (−1)
- set `BUFFLEN` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

To remove the message from the queue and discard it:

- set `DATA` or `DATALEN` to a non_NULL value (that is, not −1)
- set `BUFFLEN` to zero
- ensure that Accept Truncated Message in the policy receive attributes is selected

The message will be discarded with an AMRC_MSG_TRUNCATED warning.

If AMRC_RECEIVE_BUFF_LEN_ERR is returned, the message length value is returned in `DATALEN` (if it is non_NULL, that is, not −1), even though the completion code is MQCC_FAILED.

Note that if `DATA` is NULL (−1) and `BUFFLEN` is not zero, there is always an AMRC_RECEIVE_BUFF_LEN_ERR error.

AMHSNFL (send file)

Function to send data from a file.

```
CALL 'AMHSNFL' USING HSESSION, SENDERNAME, POLICYNAME,
                     OPTIONS, DIRNAMELEN, DIRNAME,
                     FILENAMELEN, FILENAME,
                     SNDMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  SENDERNAME    PIC X(n).
01  POLICYNAME    PIC X(n).
01  OPTIONS       PIC S9(9) BINARY.
01  DIRNAMELEN    PIC S9(9) BINARY.
01  DIRNAME       PIC X(n).
01  FILENAMELEN   PIC S9(9) BINARY.
01  FILENAME      PIC X(n).
01  SNDMSGNAME    PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SENDERNAME	The name of a sender service (input). If specified as a space or low value, the system default sender name (constant: AMSD-SND) is used.
POLICYNAME	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
OPTIONS	Reserved, must be specified as zero.
DIRNAMELEN	Reserved, must be specified as zero (input).
DIRNAME	Reserved.
FILENAMELEN	The length of the file name in bytes (input).
FILENAME	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with a receive file call (see “AMHRCFL (receive file)” on page 273 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
SNDMSGNAME	The name of the message object to be used to send the file (input). This can be used to specify the Correlation ID for example. The Correlation ID can be set from the message object using the object interface (see “Message interface functions” on page 317). If `SNDMSGNAME` is specified as a space or low value, the system default send message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

The message object is implicitly reset by this call.

The system default object is used when you set `SNDMSGNAME` as a space or low value.

AMHSNMS (send message)

Function to send a datagram (send and forget) message.

```
  CALL 'AMHSNMS' USING HSESSION, SENDER, POLICY, DATALEN, DATA,
                       SNDMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  SENDER        PIC X(n).
01  POLICY        PIC X(n).
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  SNDMSGNAME    PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SENDER	The name of a sender service (input). If specified as a space or low value, the system default sender name (constant: AMSD-SND) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
DATALEN	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (`SNDMSGNAME`) using the object interface (see “Message interface functions” on page 317).
DATA	The message data, if `DATALEN` is non-zero (input).
SNDMSGNAME	The name of a message object for the message being sent (input). If `DATALEN` is zero, the message object also holds any message data. If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHSNRQ (send request)

Function to send a request message.

```
CALL 'AMHSNRQ' USING HSESSION, SENDER, POLICY, RESPNAME, DATALEN,
                     DATA, SNDMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  SENDER        PIC X(n).
01  POLICY        PIC X(n).
01  RESPNAME      PIC X(n).
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  SNDMSGNAME    PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SENDER	The name of a sender service (input). If specified as a space or low value, the system default sender name (constant: AMSD-SND) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
RESPNAME	The name of the receiver service to which the response to this send request will be sent (input). See AMHRCRQ (receive request).
DATALEN	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (`SNDMSGNAME`) using the object interface (see “Message interface functions” on page 317).
DATA	The message data, if `DATALEN` is non-zero (input).
SNDMSGNAME	The name of a message object for the message being sent (input). If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHSNRS (send response)

Function to send a response to a request message.

```
CALL 'AMHSNRS' USING HSESSION, SENDER, POLICY, RCVMSGNAME, DATALEN,
                     DATA, SNDMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  SENDER        PIC X(n).
01  POLICY        PIC X(n).
01  RCVMSGNAME    PIC X(n).
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  SNDMSGNAME    PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SENDER	The name of the sender service (input). It must be set to the `SENDER` specified for the AMHRCRQ receive request.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
RCVMSGNAME	The name of the received message that this message is a response to (input). It must be set to the `RCVMSGNAME` specified for the AMHRCRQ receive request.
DATALEN	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (`SNDMSGNAME`) using the object interface (see "Message interface functions" on page 317).
DATA	The message data, if `DATALEN` is non-zero (input).
SNDMSGNAME	The name of a message object for the message being sent (input). If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHSB (subscribe)

Function to register a subscription with a publish/subscribe broker.

Publications matching the subscription are sent to the receiver service associated with the subscriber. By default, this has the same name as the subscriber service, with the addition of the suffix '.RECEIVER'.

Subscribing applications can exploit content based publish/subscribe by passing a filter on the AMHSUB call.

```
CALL 'AMHSB' USING HSESSION, SUBSCRIBER, POLICY, RESPNAME,
                   TOPICLEN, TOPIC, FILTERLEN, FILTER,
                   SUBMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  SUBSCRIBER    PIC X(n).
01  POLICY        PIC X(n).
01  RESPNAME      PIC X(n).
01  TOPICLEN      PIC S9(9) BINARY.
01  TOPIC         PIC X(n).
01  FILTERLEN     PIC S9(9) BINARY.
01  FILTER        PIC X(n).
01  SUBMSGNAME    PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SUBSCRIBER	The name of a subscriber service (input). If specified as a space or low value, the system default subscriber name (constant: AMSD-SUB) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
RESPNAME	The name of the receiver service to which the response to this subscribe request will be sent (input). If specified as a space or low value, no response is sent. This is not the service to which publications will be sent by the broker; they are sent to the receiver service associated with the subscriber (see above).
TOPICLEN	The length of the topic for this subscription, in bytes (input).
TOPIC	The topic for this subscription (input). Publications that match this topic, including wildcards, will be sent to the subscriber. Multiple topics can be specified in the message object (`SUBMSGNAME`) using the object interface (see "Message interface functions" on page 317).
FILTERLEN	The length in bytes of the filter (input).
FILTER	The filter to be added (input). The syntax of the filter string is described in the *WebSphere MQ Integrator Version 2.0 Programming Guide*
SUBMSGNAME	The name of a message object for the subscribe message (input). If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHTERM (terminate)

Closes the session, closes and deletes any implicitly created objects, and deletes the session. If WebSphere MQ is the transaction coordinator, any outstanding units of work are committed (if the application terminates without an AMHTERM call being issued, any outstanding units of work are backed out).

```
CALL 'AMHTERM' USING HSESSION, POLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  POLICY        PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHUN (unsubscribe)

Function to remove a subscription from a publish/subscribe broker.

```
CALL 'AMHUN' USING HSESSION, SUBSCRIBER, POLICY, RESPNAME,
                   TOPICLEN, TOPIC, FILTERLEN, FILTER,
                   UNSUBMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESSION      PIC S9(9) BINARY.
01  SUBSCRIBER    PIC X(n).
01  POLICY        PIC X(n).
01  RESPNAME      PIC X(n).
01  TOPICLEN      PIC S9(9) BINARY.
01  TOPIC         PIC X(n).
01  FILTERLEN     PIC S9(9) BINARY.
01  FILTER        PIC X(n).
01  UNSUBMSGNAME  PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SUBSCRIBER	The name of a subscriber service (input). If specified as a space or low value, the system default subscriber name (constant: AMSD-SUB) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
RESPNAME	The name of the receiver service to which the response to this unsubscribe request will be sent (input).
TOPICLEN	The length of the topic, in bytes (input).
TOPIC	The topic that identifies the subscription which is to be removed (input). Multiple topics can be specified in the message object (`UNSUBMSGNAME`) using the object interface (see “Message interface functions” on page 317). To deregister all topics, a policy providing this option must be specified (this is not the default policy). Otherwise, to remove a previous subscription the topic information specified must match that specified on the relevant AMHSB subscribe request.
FILTERLEN	The length in bytes of the filter (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
FILTER	The filter that identifies the subscription to be removed (input). The syntax of the filter string is described in the *WebSphere MQ Integrator Version 2.0 Programming Guide*
UNSUBMSGNAME	The name of a message object for the unsubscribe message (input). If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

To successfully remove a previous subscription, you must ensure that the topic, filter, and subscriber queue information exactly matches that used on the original subscribe request.

Chapter 12. COBOL object interface overview

This chapter contains an overview of the structure of the COBOL object interface. Use it to find out what functions are available in this interface.

The object interface provides sets of interface functions for each of the following objects:

Session	page 290
Message	page 292
Sender	page 295
Receiver	page 296
Distribution list	page 297
Publisher	page 298
Subscriber	page 299
Policy	page 300

These interface functions are invoked as necessary by the high-level functions. They are made available to the application programmer through this object-style interface to provide additional function where needed. An application program can mix high-level functions and object-interface functions as required.

Details of the interface functions for each object are given in the following pages. Follow the page references to see the detailed descriptions of each function.

Details of the object interface functions used by each high-level function are given on page 301.

Session interface functions

The session object creates and manages all other objects, and provides the scope for a unit of work.

Session management

Functions to create, open, close, and delete a session object.

AMSECR (create)	page 306
AMSEOP (open)	page 316
AMSECL (close)	page 305
AMSEDL (delete)	page 310

Create objects

Functions to create message, sender, receiver, distribution list, publisher, subscriber, and policy objects. Handles to these objects are returned by these functions.

AMSECRMS (create message)	page 307
AMSECRSN (create sender)	page 309
AMSECRRC (create receiver)	page 308
AMSECRDL (create distribution list)	page 306
AMSECRPB (create publisher)	page 308
AMSECRSB (create subscriber)	page 309
AMSECRPO (create policy)	page 307

Get object handles

Functions to get the handles for a message, sender, receiver, distribution list, publisher, subscriber, and policy objects with a specified name (needed if the objects were created implicitly by the high-level interface).

AMSEGHMS (get message handle)	page 313
AMSEGHSN (get sender handle)	page 315
AMSEGHRC (get receiver handle)	page 314
AMSEGHDL (get distribution list handle)	page 312
AMSEGHPB (get publisher handle)	page 314
AMSEGHSB (get subscriber handle)	page 315
AMSEGHPO (get policy handle)	page 314

Delete objects

Functions to delete message, sender, receiver, distribution list, publisher, subscriber, and policy objects.

AMSEDLMS (delete message)	page 310
AMSEDLSN (delete sender)	page 312
AMSEDLRC (delete receiver)	page 311
AMSEDLDL (delete distribution list)	page 310
AMSEDLPB (delete publisher)	page 311
AMSEDLSB (delete subscriber)	page 312
AMSEDLPO (delete policy)	page 311

Transactional processing

Functions to begin, commit, and rollback a unit of work.

AMSEBG (begin)	page 304
AMSECM (commit)	page 305
AMSERB (rollback)	page 316

Error handling

Functions to clear the error codes, and return the completion and reason codes for the last error associated with the session object.

AMSECLEC (clear error codes)	page 304
AMSEGTLE (get last error codes)	page 313

Message interface functions

A message object encapsulates a WebSphere MQ message descriptor (MQMD) structure. It also contains the message data if this is not passed as a separate parameter.

Get values

Functions to get the coded character set ID, correlation ID, encoding, format, group status, message ID, name, report code, and type of the message object.

AMMSGTCC (get CCSID)	page 321
AMMSGTCI (get correl ID)	page 322
AMMSGELC (get element CCSID)	page 321
AMMSGTEN (get encoding)	page 324
AMMSGTFO (get format)	page 325
AMMSGTGS (get group status)	page 326
AMMSGTMI (get message ID)	page 327
AMMSGTNA (get name)	page 327
AMMSGTRC (get report code)	page 329
AMMSGTTY (get type)	page 330

Set values

Functions to set the coded character set ID, correlation ID, encoding, format, and group status of the message object.

AMMSSTCC (set CCSID)	page 332
AMMSSTCI (set correl ID)	page 332
AMMSSELC (set element CCSID)	page 332
AMMSSTEN (set encoding)	page 333
AMMSSTFO (set format)	page 334
AMMSSTGS (set group status)	page 334
AMMSSTRC (set report code)	page 335
AMMSSTTY (set type)	page 335

Reset values

Function to reset the message object to the state it had when first created.

AMMSRS (reset)	page 331

Read and write data

Functions to get the length of the data, get and set the data offset, and read or write byte data to or from the message object at the current offset.

AMMSGTDL (get data length)	page 322
AMMSGTDO (get data offset)	page 322
AMMSSTDO (set data offset)	page 333
AMMSREBY (read bytes)	page 331
AMMSWRBY (write bytes)	page 336

Publish/subscribe topics

Functions to manipulate the topics in a publish/subscribe message.

AMMSADTO (add topic)	page 319
AMMSDETO (delete topic)	page 321
AMMSGTTO (get topic)	page 329
AMMSGTTC (get topic count)	page 330

Publish/subscribe filters

Functions to manipulate the filters in a publish/subscribe message.

AMMSADFI (add filter)	page 318
AMMSDEFI (delete filter)	page 320
AMMSGTFI (get filter)	page 325
AMMSGTFC(get filter count)	page 324

Publish/subscribe name/value elements

Functions to manipulate the name/value elements in a publish/subscribe message.

AMMSADEL (add element)	page 318
AMMSDEEL (delete element)	page 319
AMMSGTEL (get element)	page 323
AMMSGTEC (get element count)	page 323
AMMSDENE (delete named element)	page 320
AMMSGTNE (get named element)	page 328
AMMSGTNC (get named element count)	page 328

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the message.

COBOL object interface overview

Sender interface functions

A sender object encapsulates a WebSphere MQ object descriptor (MQOD) structure for sending a message.

Open and close

Functions to open and close the sender service.

AMSNOP (open)	page 340
AMSNCL (close)	page 338

Send

Function to send a message.

AMSNSN (send)	page 340
AMSNSNFL(send file)	page 341

Get values

Functions to get the coded character set ID, encoding, and name of the sender service.

AMSNGTCC (get CCSID)	page 338
AMSNGTEN (get encoding)	page 338
AMSNGTNA (get name)	page 339

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the sender service.

AMSNCLEC (clear error codes)	page 337
AMSNGTLE (get last error)	page 339

Receiver interface functions

A receiver object encapsulates a WebSphere MQ object descriptor (MQOD) structure for receiving a message.

Open and close

Functions to open and close the receiver service.

AMRCOP (open)	page 349
AMRCCL (close)	page 347

Receive and browse

Functions to receive or browse a message.

AMRCRC (receive)	page 350
AMRCRCFL (receive file)	page 351
AMRCBR (browse)	page 343
AMRCBRSE (browse selection message)	page 345

Get values

Functions to get the definition type, name, and queue name of the receiver service.

AMRCGTDT (get definition type)	page 347
AMRCGTNA (get name)	page 348
AMRCGTQN (get queue name)	page 349

Set values

Function to set the queue name of the receiver service.

AMRCSTQN (set queue name)	page 352

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the receiver service.

AMRCCLEC (clear error codes)	page 346
AMRCGTLE (get last error)	page 348

Distribution list interface functions

A distribution list object encapsulates a list of sender services.

Open and close

Functions to open and close the distribution list service.

AMDLOP (open)	page 355
AMDLCL (close)	page 353

Send

Function to send a message to the distribution list.

AMDLSN (send)	page 356
AMDLSNFL (send file)	page 356

Get values

Functions to get the name of the distribution list service, a count of the sender services in the list, and a sender service handle.

AMDLGTNA (get name)	page 354
AMDLGTSC (get sender count)	page 354
AMDLGTSH (get sender handle)	page 355

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the distribution list.

AMDLCLEC (clear error codes)	page 353
AMDLGTLE (get last error)	page 353

Publisher interface functions

A publisher object encapsulates a sender service. It provides support for publishing messages to a publish/subscribe broker.

Open and close

Functions to open and close the publisher service.

AMPBOP (open)	page 360
AMPBCL (close)	page 358

Publish

Function to publish a message.

AMPBPB (publish)	page 361

Get values

Functions to get the coded character set ID, encoding, and name of the publisher service.

AMPBGTCC (get CCSID)	page 358
AMPBGTEN (get encoding)	page 359
AMPBGTNA (get name)	page 360

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the publisher.

AMPBCLEC (clear error codes)	page 358
AMPBGTLE (get last error)	page 359

Subscriber interface functions

A subscriber object encapsulates both a sender service and a receiver service. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

Open and close

Functions to open and close the subscriber service.

AMSBOP (open)	page 366
AMSBCL (close)	page 362

Broker messages

Functions to subscribe to a broker, remove a subscription, and receive publications from the broker.

AMSBSB (subscribe)	page 367
AMSBUN (unsubscribe)	page 368
AMSBRC (receive)	page 366

Get values

Functions to get the coded character set ID, definition type, encoding, name, and queue name of the subscriber service.

AMSBGTCC (get CCSID)	page 363
AMSBGTDT (get definition type)	page 363
AMSBGTEN (get encoding)	page 364
AMSBGTNA (get name)	page 365
AMSBGTQN (get queue name)	page 365

Set value

Function to set the queue name of the subscriber service.

AMSBSTQN (set queue name)	page 367

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the receiver.

AMSBCLEC (clear error codes)	page 362
AMSBGTLE (get last error)	page 364

Policy interface functions

A policy object encapsulates details of how the message is handled (such as priority, persistence, and whether it is included in a unit of work).

Get values

Functions to get the name of the policy, and the wait time set in the policy.

AMPOGTNA (get name)	page 370
AMPOGTWT (get wait time)	page 370

Set value

Function to set the wait time for a receive using the policy.

AMPOSTWT (set wait time)	page 371

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the policy.

AMPOCLEC (clear error codes)	page 369
AMPOGTLE (get last error)	page 369

High-level functions

Each high-level function described in Chapter 11, “The COBOL high-level interface” on page 263 calls a number of the object interface functions, as shown below.

Table 4. Object interface calls used by the high-level functions

High-level function	Equivalent object interface calls
AMHBACK (backout)	AMSECRPO / AMSEGHPO AMSERB
AMHBEGIN (begin)	AMSECRPO / AMSEGHPO AMSEBG
AMHBRMS (browse message)	AMSECRRC / AMSEGHRC AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMRCBRSE
AMHCMIT (commit)	AMSECRPO / AMSEGHPO AMSECM
AMHINIT (initialize)	AMSECR AMSEOP
AMHTERM (terminate)	AMSECL AMSEDL
AMHSNMS (send message) AMHSNRQ (send request) AMHSNRS (send response)	AMSECRSN / AMSEGHSN AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMSNSN
AMHRCMS (receive message) AMHRCRQ (receive request)	AMSECRRC / AMSEGHRC AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMRCRC
AMHSNFL (send file)	AMSECRSN / AMSEGHSN AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMSNSNFL
AMHRCFL (receive file)	AMSECRRC / AMSEGHRC AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMRCRCFL
AMHPB (publish)	AMSECRPB / AMSEGHPB AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMPBPB
AMHSB (subscribe)	AMSECRSB / AMSEGHSB AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMSBSB
AMHUN (unsubscribe)	AMSECRSB / AMSEGHSB AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMSBUN
AMHRCPB (receive publication)	AMSECRSB / AMSEGHSB AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMSBRC

If an object already exists, the appropriate call to get its handle is used instead of calling the create function again. For example, if the policy object exists, AMSEGHPO (get policy handle) is used instead of AMSECRPO (create policy).

Chapter 13. COBOL object interface reference

In the following sections the COBOL object interface functions are listed by the object they refer to:

Within each section the functions are listed in alphabetical order.

Note that all functions return a completion code (COMPCODE) and a reason code (REASON). The completion code can take one of the following values:

AMCC-OK	Function completed successfully
AMCC-WARNING	Function completed with a warning
AMCC-FAILED	An error occurred during processing

If the completion code returns warning or failed, the reason code identifies the reason for the error or warning (see Appendix A, "Reason codes and LDAP error codes" on page 537).

Most functions require a handle to the object they reference. If this handle is not valid, the results are unpredictable.

Session interface functions

A *session* object provides the scope for a unit of work and creates and manages all other objects, including at least one connection object. Each (WebSphere MQ) connection object encapsulates a single WebSphere MQ queue manager connection. The session object definition specifying the required queue manager connection can be provided by a repository policy definition and the local host file, or the local host file only which by default will name a single local queue manager with no repository. (Under CICS, there can be only one queue manager connected to a given CICS system, so in this case the local host file is irrelevant.) The session, when deleted, is responsible for releasing memory by closing and deleting all other objects that it manages.

Note that you should not mix WebSphere MQ MQCONN or MQDISC requests on the same thread as AMI calls, otherwise premature disconnection might occur.

AMSEBG (begin)

Begins a unit of work, allowing an AMI application to take advantage of the resource coordination provided in WebSphere MQ. The unit of work can subsequently be committed by AMSECM, or backed out by AMSERB. It should be used only when WebSphere MQ is the transaction coordinator. If an external transaction coordinator (for example, CICS or Tuxedo) is being used, the API of the external coordinator should be used instead.

```
CALL 'AMSEBG' USING HSESS, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECLEC (clear error codes)

Clears the error codes in the session object.

```
CALL 'AMSECLEC' USING HSESS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECL (close)

Closes the session object and all open objects owned by the session, and disconnects from the underlying message transport (WebSphere MQ).

```
CALL 'AMSECL' USING HSESS, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS          PIC S9(9) BINARY.
01  HPOLICY        PIC S9(9) BINARY.
01  COMPCODE       PIC S9(9) BINARY.
01  REASON         PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECM (commit)

Commits a unit of work that was started by AMSEBG, or by sending or receiving a message under syncpoint control as defined in the policy options for the send or receive request.

```
CALL 'AMSECM' USING HSESS, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS          PIC S9(9) BINARY.
01  HPOLICY        PIC S9(9) BINARY.
01  COMPCODE       PIC S9(9) BINARY.
01  REASON         PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECR (create)

Creates the session and system default objects. AMSECR returns the handle of the session object. This must be specified by other session function calls.

```
CALL 'AMSECR' USING NAME, HSESS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  NAME          PIC X(n).
01  HSESS         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

NAME	An optional session name that can be used to identify the application from which a message is sent (input).
HSESS	The handle of the session object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECRDL (create distribution list)

Creates a distribution list object. A distribution list handle is returned.

```
CALL 'AMSECRDL' USING HSESS, NAME, HDISTLIST, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HDISTLIST     PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the distribution list (input). This must match the name of a distribution list defined in the repository.
HDISTLIST	The handle of the distribution list object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECRMS (create message)

Creates a message object. A message handle is returned.

```
CALL 'AMSECRMS' USING HSESS, NAME, HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS          PIC S9(9) BINARY.
01  NAME           PIC X(n).
01  HMSG           PIC S9(9) BINARY.
01  COMPCODE       PIC S9(9) BINARY.
01  REASON         PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the message (input). This can be any name that is meaningful to the application. It is specified so that this message object can be used with the high-level interface.
HMSG	The handle of the message object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECRPO (create policy)

Creates a policy object. A policy handle is returned.

```
CALL 'AMSECRPO' USING HSESS, NAME, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS          PIC S9(9) BINARY.
01  NAME           PIC X(n).
01  HPOLICY        PIC S9(9) BINARY.
01  COMPCODE       PIC S9(9) BINARY.
01  REASON         PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the policy (input). If it matches a policy defined in the repository, the policy will be created using the repository definition, otherwise it will be created with default values. If a repository is being used and the named policy is not found in the repository, a completion code of AMCC-WARNING is returned with a reason code of AMRC-POLICY-NOT-IN-REPOS.
HPOLICY	The handle of the policy object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECRPB (create publisher)

Creates a publisher object. A publisher handle is returned.

```
CALL 'AMSECRPB' USING HSESS, NAME, HPUBLISHER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HPUBLISHER    PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the publisher (input). If it matches a publisher defined in the repository, the publisher will be created using the repository definition, otherwise it will be created with default values (that is, with a sender service name that matches the publisher name). If a repository is being used and the named publisher is not found in the repository, a completion code of AMCC-WARNING is returned with a reason code of AMRC-PUBLISHER-NOT-IN-REPOS.
HPUBLISHER	The handle of the publisher object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECRRC (create receiver)

Creates a receiver service object. A receiver handle is returned.

```
CALL 'AMSECRRC' USING HSESS, NAME, HRECEIVER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HRECEIVER     PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the receiver service (input). If it matches a receiver defined in the repository, the receiver will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the receiver name). If a repository is being used and the named receiver is not found in the repository, a completion code of AMCC-WARNING is returned with a reason code of AMRC-RECEIVER-NOT-IN-REPOS.
HRECEIVER	The handle of the receiver object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECRSN (create sender)

Creates a sender service object. A sender handle is returned.

```
CALL 'AMSECRSN' USING HSESS, NAME, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HSENDER       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the sender service (input). If it matches a sender defined in the repository, the sender will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the sender name). If a repository is being used and the named sender is not found in the repository, a completion code of AMCC-WARNING is returned with a reason code of AMRC-SENDER-NOT-IN-REPOS.
HSENDER	The handle of the sender object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSECRSB (create subscriber)

Creates a subscriber object. A subscriber handle is returned.

```
CALL 'AMSECRSB' USING HSESS, NAME, HSUBSCRIBER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the subscriber (input). If it matches a subscriber defined in the repository, the subscriber will be created using the repository definition, otherwise it will be created with default values (that is, with a sender service name that matches the subscriber name, and a receiver service name that is the same with the addition of the suffix '.RECEIVER'). If a repository is being used and the named subscriber is not found in the repository, a completion code of AMCC-WARNING is returned with a reason code of AMRC-SUBSCRIBER-NOT-IN-REPOS.
HSUBSCRIBER	The handle of the subscriber object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEDL (delete)

Deletes the session object. Performs an implicit close if the session is open. This closes and deletes the session and all objects owned by it.

```
CALL 'AMSEDL' USING HSESS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS          PIC S9(9) BINARY.
01  COMPCODE       PIC S9(9) BINARY.
01  REASON         PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEDLDL (delete distribution list)

Deletes a distribution list object, and performs an implicit close if the distribution list is open.

```
CALL 'AMSEDLDL' USING HSESS, HDISTLIST, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS          PIC S9(9) BINARY.
01  HDISTLIST      PIC S9(9) BINARY.
01  COMPCODE       PIC S9(9) BINARY.
01  REASON         PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HDISTLIST	The distribution list handle returned by AMSECRDL (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEDLMS (delete message)

Deletes a message object.

```
CALL 'AMSEDLMS' USING HSESS, HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS          PIC S9(9) BINARY.
01  HMSG           PIC S9(9) BINARY.
01  COMPCODE       PIC S9(9) BINARY.
01  REASON         PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HMSG	The message handle returned by AMSECRMS (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEDLPO (delete policy)

Deletes a policy object.

```
CALL 'AMSEDLPO' USING HSESS, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HPOLICY	The policy handle returned by AMSECRPO (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEDLPB (delete publisher)

Deletes a publisher object, and performs an implicit close if the publisher is open.

```
CALL 'AMSEDLPB' USING HSESS, HPUBLISHER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  HPUBLISHER    PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HPUBLISHER	The publisher handle returned by AMSECRPB (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEDLRC (delete receiver)

Deletes a receiver object, and performs an implicit close if the receiver is open.

```
CALL 'AMSEDLRC' USING HSESS, HRECEIVER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  HRECEIVER     PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HRECEIVER	The receiver handle returned by AMSECRRC (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEDLSN (delete sender)

Deletes a sender object, and performs an implicit close if the sender is open.

```
CALL 'AMSEDLSN' USING HSESS, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  HSENDER       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HSENDER	The sender handle returned by AMSECRSN (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEDLSB (delete subscriber)

Deletes a subscriber object, and performs an implicit close if the subscriber is open.

```
CALL 'AMSEDLSB' USING HSESS, HSUBSCRIBER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEGHDL (get distribution list handle)

Returns the handle of the distribution list object with the specified name.

```
CALL 'AMSEGHDL' USING HSESS, NAME, HDISTLIST, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HDISTLIST     PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the distribution list (input).
HDISTLIST	The handle of the distribution list object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEGTLE (get last error codes)

Gets the information (completion and reason codes) from the last error for the session.

```
CALL 'AMSEGTLE' USING HSESS, BUFFLEN, STRINGLEN, ERRORTEXT,
                      REASON2, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  STRINGLEN     PIC S9(9) BINARY.
01  ERRORTEXT     PIC X(n).
01  REASON2       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
BUFFLEN	Reserved, must be zero (input).
STRINGLEN	Reserved (output).
ERRORTEXT	Reserved (output).
REASON2	A secondary reason code (output). If `REASON` indicates AMRC-TRANSPORT-WARNING or AMRC-TRANSPORT-ERR, `REASON2` gives a WebSphere MQ reason code.
COMPCODE	Completion code (output).
REASON	Reason code (output). A value of AMRC-SESSION-HANDLE-ERR indicates that the AMSEGTLE function call has itself detected an error and failed.

AMSEGHMS (get message handle)

Returns the handle of the message object with the specified name.

```
CALL 'AMSEGHMS' USING HSESS, NAME, HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HMSG          PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the message (input).
HMSG	The handle of the message object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEGHPO (get policy handle)

Returns the handle of the policy object with the specified name.

```
CALL 'AMSEGHPO' USING HSESS, NAME, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the policy (input).
HPOLICY	The handle of the policy object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEGHPB (get publisher handle)

Returns the handle of the publisher object with the specified name.

```
CALL 'AMSEGHPB' USING HSESS, NAME, HPUBLISHER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HPUBLISHER    PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the publisher (input).
HPUBLISHER	The handle of the publisher object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEGHRC (get receiver handle)

Returns the handle of the receiver service object with the specified name.

```
CALL 'AMSEGHRC' USING HSESS, NAME, HRECEIVER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HRECEIVER     PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the receiver (input).
HRECEIVER	The handle of the receiver object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEGHSN (get sender handle)

Returns the handle of the sender service object with the specified name.

```
CALL 'AMSEGHSN' USING HSESS, NAME, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HSENDER       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the sender (input).
HSENDER	The handle of the sender object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEGHSB (get subscriber handle)

Returns the handle of the subscriber object with the specified name.

```
CALL 'AMSEGHSB' USING HSESS, NAME, HSUBSCRIBER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
NAME	The name of the subscriber (input).
HSUBSCRIBER	The handle of the subscriber object (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSEOP (open)

Opens the session object using the specified policy options. The policy, together with the local host file, provides the connection definition that enables the connection object to be created. The specified library is loaded and initialized. (Because client connections are not supported on z/OS, programs running on z/OS must use a local queue manager). The connection to the underlying message transport (WebSphere MQ) is then opened.

```
CALL 'AMSEOP' USING HSESS, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS          PIC S9(9) BINARY.
01  HPOLICY        PIC S9(9) BINARY.
01  COMPCODE       PIC S9(9) BINARY.
01  REASON         PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSERB (rollback)

Rolls back a unit of work.

```
CALL 'AMSERB' USING HSESS, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS          PIC S9(9) BINARY.
01  HPOLICY        PIC S9(9) BINARY.
01  COMPCODE       PIC S9(9) BINARY.
01  REASON         PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECR (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Message interface functions

A *message* object encapsulates a WebSphere MQ message descriptor (MQMD), and name/value elements such as the topic data for publish/subscribe messages. It can also contain the message data, or this can be passed as a separate parameter.

A name/value element in a message object is held in an AMELEM structure. See "Using name/value elements" on page 255 for details.

The initial state of the message object is:

CCSID	default queue manager CCSID
CORRELATIONID	all zeros
DATALENGTH	zero
DATAOFFSET	zero
ELEMENTCOUNT	zero
ENCODING	AMENC-NATIVE
FORMAT	AMFMT-STRING
GROUPSTATUS	AMGRP-MSG-NOT-IN-GROUP
TOPICCOUNT	zero

When a message object is used to send a message, it will not normally be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see AMMSRS on page 331) and rebuild it each time.

Note that the following calls are valid only after a session has been opened with an **AMSEOP** call or after you have explicitly set the element CCSID with an **AMMSSELC** call:

AMMSADEL (add element)	page 318
AMMSDEEL (delete element)	page 319
AMMSGTEL (get element)	page 323
AMMSGTEC (get element count)	page 323
AMMSDENE (delete named element)	page 320
AMMSGTNE (get named element)	page 328
AMMSGTNC (get named element count)	page 328
AMMSADTO (add topic)	page 319
AMMSDETO (delete topic)	page 321
AMMSGTTO (get topic)	page 329
AMMSGTTC (get topic count)	page 330

AMMSADEL (add element)

Adds a name/value element to a message (such as a publish/subscribe message).

```
CALL 'AMMSADEL' USING HMSG, AMELEM, OPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  AMELEM.
    COPY AMTELEMV.
01  OPTIONS       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
AMELEM	An AMELEM element structure, which specifies the element to be added (input). It will not replace an existing element with the same name.
OPTIONS	Reserved, must be set to zero (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSADFI (add filter)

Adds a filter to a subscribe or unsubscribe request message.

```
CALL 'AMMSADFI' USING HMSG, FILTERLEN, TOPIC, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  FILTERLEN     PIC S9(9) BINARY,
01  FILTER        PIC X(n),
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
FILTERLEN	The length in bytes of the filter (input). A value of AMLEN-NULL-TERM specifies that the string is null terminated.
FILTER	The filter to be added (input). The syntax of the filter string is described in the *WebSphere MQ Integrator Version 2.0 Programming Guide*.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSADTO (add topic)

Adds a topic to a publish/subscribe message.

```
CALL 'AMMSADTO' USING HMSG, TOPICLEN, TOPIC, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG         PIC S9(9) BINARY.
01  TOPICLEN     PIC S9(9) BINARY.
01  TOPIC        PIC X(n).
01  COMPCODE     PIC S9(9) BINARY.
01  REASON       PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
TOPICLEN	The length in bytes of the topic (input).
TOPIC	The topic to be added (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSCLEC (clear error codes)

Clears the error codes in the message object.

```
CALL 'AMMSCLEC' USING HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG         PIC S9(9) BINARY.
01  COMPCODE     PIC S9(9) BINARY.
01  REASON       PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSDEEL (delete element)

Deletes an element with the specified index from a message (such as a publish/subscribe message). Indexing is within all elements of the message, and might include topics or filters (which are specialized elements).

```
CALL 'AMMSDEEL' USING HMSG, ELEMINDEX, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG         PIC S9(9) BINARY.
01  ELEMINDEX    PIC S9(9) BINARY.
01  COMPCODE     PIC S9(9) BINARY.
01  REASON       PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
ELEMINDEX	The index of the required element in the message, starting from zero (input). On completion, elements with higher `ELEMINDEX` values than that specified will have their index value reduced by one. Use AMMSGTEC to get the number of elements in the message.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSDEFI (delete filter)

Deletes a filter from a subscribe or unsubscribe message at the specified index. Indexing is within all filters.

```
CALL 'AMMSDEFI' USING HMSG, FILTERINDEX, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  FILTERINDEX   PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
FILTERINDEX	The index of the required filter in the message, starting from zero (input). **AMMSGTFI** gets the number of filters in the message.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSDENE (delete named element)

Deletes a named element from a message (such as a publish/subscribe message), at the specified index. Indexing is within all elements that share the same name.

```
CALL 'AMMSDENE' USING HMSG, NAMEINDEX, NAMELEN, NAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  NAMEINDEX     PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
NAMEINDEX	The index of the required named element in the message (input). Specifying an index of zero deletes the *first* element with the specified name. On completion, elements with higher `NAMEINDEX` values than that specified will have their index value reduced by one. Use AMMSGTNC to get the number of elements in the message with the specified name.
NAMELEN	The length of the element name, in bytes (input).
NAME	The name of the element to be deleted (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSDETO (delete topic)

Deletes a topic from a publish/subscribe message, at the specified index. Indexing is within all topics in the message.

```
CALL 'AMMSDETO' USING HMSG, TOPICINDEX, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  TOPICINDEX    PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
TOPICINDEX	The index of the required topic in the message, starting from zero (input). On completion, topics with higher `TOPICINDEX` values than that specified will have their index value reduced by one. Use AMMSGTTC to get the number of topics in the message.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGELC (get element CCSID)

Gets the message element CCSID. This is the coded character set identifier used for passing message element data (including topic and filter data) to or from an application.

```
CALL 'AMMSGELCC' USING HMSG, ELEMENTCCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  ELEMENTCCSID  PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
ELEMENTCCSID	The element coded character set identifier (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTCC (get CCSID)

Gets the coded character set identifier of the message.

```
CALL 'AMMSGTCC' USING HMSG, CCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  CCSID         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
CCSID	The coded character set identifier (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTCI (get correl ID)

Gets the correlation identifier of the message.

```
CALL 'AMMSGTCI' USING HMSG, BUFFLEN, CORRELIDLEN, CORRELID,
                      COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG         PIC S9(9) BINARY.
01  BUFFLEN      PIC S9(9) BINARY.
01  CORRELIDLEN  PIC S9(9) BINARY.
01  CORRELID     PIC X(n).
01  COMPCODE     PIC S9(9) BINARY.
01  REASON       PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
BUFFLEN	The length in bytes of a buffer in which the correlation identifier is returned (input).
CORRELIDLEN	The length of the correlation identifier, in bytes (output).
CORRELID	The correlation identifier (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTDL (get data length)

Gets the length of the message data in the message object.

```
CALL 'AMMSGTDL' USING HMSG, LENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG         PIC S9(9) BINARY.
01  LENGTH       PIC S9(9) BINARY.
01  COMPCODE     PIC S9(9) BINARY.
01  REASON       PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
LENGTH	The length of the message data, in bytes (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTDO (get data offset)

Gets the current offset in the message data for reading or writing data bytes.

```
CALL 'AMMSGTDO' USING HMSG, OFFSET, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG         PIC S9(9) BINARY.
01  OFFSET       PIC S9(9) BINARY.
01  COMPCODE     PIC S9(9) BINARY.
01  REASON       PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
OFFSET	The byte offset in the message data (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTEL (get element)

Gets an element from a message.

```
CALL 'AMMSGTEL' USING HMSG, ELEMINDEX, ELEM, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  ELEMINDEX     PIC S9(9) BINARY.
01  ELEM.
    COPY AMTELEMV.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
ELEMINDEX	The index of the required element in the message, starting from zero (input). Use AMMSGTEC to get the number of elements in the message.
ELEM	The selected element in the message (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTEC (get element count)

Gets the total number of elements in a message.

```
CALL 'AMMSGTEC' USING HMSG, COUNT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  COUNT         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
COUNT	The number of elements in the message (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTEN (get encoding)

Gets the value used to encode numeric data types for the message.

```
CALL 'AMMSGTEN' USING HMSG, ENCODING, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  ENCODING      PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

ENCODING The encoding of the message (output). The following values can be returned:

```
AMENC-NATIVE
AMENC-NORMAL
AMENC-NORMAL-FLOAT-390
AMENC-REVERSED
AMENC-REVERSED-FLOAT-390
AMENC-UNDEFINED
```

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTFC (get filter count)

Gets the total number of filters in a publish/subscribe message.

```
CALL 'AMMSGTFC' USING HMSG, COUNT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  COUNT         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

COUNT The number of filters (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTFI (get filter)

Get a filter from a publish/subscribe message at the specified index. Indexing is within all filters.

```
CALL 'AMMSGTFI' USING HMSG, INDEX, BUFFLEN, FILTERLEN,
                      FILTER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  INDEX         PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  FILTERLEN     PIC S9(9) BINARY.
01  FILTER        PIC X(N),
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
INDEX	The index of the required filter in the message (input). Specifying an index of zero returns the first filter. `AMMSGTFC` gets the number of filters in the message.
BUFFLEN	The length in bytes of a buffer in which the filter is returned (input).
FILTERLEN	The length of the filter, in bytes (output).
FILTER	The filter (output)
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTFO (get format)

Gets the format of the message.

```
CALL 'AMMSGTFO' USING HMSG, BUFFLEN, FORMATLEN, FORMAT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  FORMATLEN     PIC S9(9) BINARY.
01  FORMAT        PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
BUFFLEN	The length in bytes of a buffer in which the format is returned (input).
FORMATLEN	The length of the format, in bytes (output).

FORMAT	The format of the message (output). The values that can be returned include the following:
	`AMFMT-NONE`
	`AMFMT-STRING`
	`AMFMT-RF-HEADER`

COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTGS (get group status)

Gets the group status of the message. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group.

```
CALL 'AMMSGTGS' USING HMSG, STATUS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  STATUS        PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
STATUS	The group status (output). It can take one of the following values: AMGRP-MSG-NOT-IN-GROUP AMGRP-FIRST-MSG-IN-GROUP AMGRP-MIDDLE-MSG-IN-GROUP AMGRP-LAST-MSG-IN-GROUP AMGRP-ONLY-MSG-IN-GROUP Alternatively, bitwise tests can be performed using the constants: AMGF-IN-GROUP AMGF-FIRST AMGF-LAST
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTLE (get last error)

Gets the information (completion and reason codes) from the last error for the message object.

```
CALL 'AMMSGTLE' USING HSESS, BUFFLEN, STRINGLEN, ERRORTEXT,
                      REASON2, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSESS         PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  STRINGLEN     PIC S9(9) BINARY.
01  ERRORTEXT     PIC X(n).
01  REASON2       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSESS	The session handle returned by AMSECRMS (input).
BUFFLEN	Reserved, must be zero (input).
STRINGLEN	Reserved (output).
ERRORTEXT	Reserved (output).
REASON2	A secondary reason code (output). If `REASON` indicates AMRC-TRANSPORT-WARNING or AMRC-TRANSPORT-ERR, `REASON2` gives a WebSphere MQ reason code.
COMPCODE	Completion code (output).
REASON	Reason code (output). A value of AMRC-MSG-HANDLE-ERR indicates that the AMMSGTLE function call has itself detected an error and failed.

AMMSGTMI (get message ID)

Gets the message identifier.

```
CALL 'AMMSGTMI' USING HMSG, BUFFLEN, MSGIDLEN, MSGID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  MSGIDLEN      PIC S9(9) BINARY.
01  MSGID         PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
BUFFLEN	The length in bytes of a buffer in which the message identifier is returned (input).
MSGIDLEN	The length of the message identifier, in bytes (output).
MSGID	The message identifier (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTNA (get name)

Gets the name of the message object.

```
CALL 'AMMSGTNA' USING HMSG, BUFFLEN, NAMELEN, NAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
BUFFLEN	The length in bytes of a buffer in which the name is returned (input).
NAMELEN	The length of the name, in bytes (output).
NAME	The message object name (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTNE (get named element)

Gets a named element from a message (such as a publish/subscribe message).

```
  CALL 'AMMSGTNE' USING HMSG, NAMEINDEX, NAMELEN, NAME, ELEM
                        COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  NAMEINDEX     PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  ELEM.
    COPY AMTELEMV.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
NAMEINDEX	The index of the required named element in the message (input). Specifying an index of zero returns the first element with the specified name. Use AMMSGTNC to get the number of elements in the message with the specified name.
NAMELEN	The length of the element name, in bytes (input).
NAME	The element name (input).
ELEM	The selected named element in the message (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTNC (get named element count)

Gets the number of elements in a message with a specified name.

```
  CALL 'AMMSGTNC' USING HMSG, NAMELEN, NAME, COUNT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  COUNT         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
NAMELEN	The length of the element name, in bytes (input).
NAME	The specified element name (input).
COUNT	The number of elements in the message with the specified name (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTRC (get report code)

Gets the feedback code from a message of type AMMT-REPORT. If the message type is not AMMT-REPORT, error code AMRC-MSG-TYPE-NOT-REPORT will be returned.

```
CALL 'AMMSGTRC' USING HMSG, REPORTCODE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  REPORTCODE    PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).

REPORTCODE	The feedback code (output). In addition to application defined values, the following can be returned:
	`AMFB-NONE`
	`AMFB-CODE-EXPIRATION`
	`AMFB-CODE-COA`
	`AMFB-CODE-COD`
	Error code AMRC_MSG_TYPE_NOT_REPORT may be issued.

COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTTO (get topic)

Gets a topic from a publish/subscribe message, at the specified index. Indexing is within all topics.

```
CALL 'AMMSGTTO' USING HMSG, TOPICINDEX, BUFFLEN, TOPICLEN, TOPIC,
                      COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  TOPICINDEX    PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  TOPICLEN      PIC S9(9) BINARY.
01  TOPIC         PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
TOPICINDEX	The index of the required topic in the message (input). Specifying an index of zero returns the first topic. Use AMMSGTTC to get the number of topics in the message.
BUFFLEN	The length in bytes of a buffer in which the topic is returned (input). If `BUFFLEN` is specified as zero, only the topic length is returned (in `TOPICLEN`), not the topic itself.
TOPICLEN	The length of the topic, in bytes (output).
TOPIC	The topic (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTTC (get topic count)

Gets the total number of topics in a publish/subscribe message.

```
CALL 'AMMSGTTC' USING HMSG, COUNT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  COUNT         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
COUNT	The number of topics (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSGTTY (get type)

Gets the type from a message.

```
CALL 'AMMSGTTY' USING HMSG, TYPE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  TYPE          PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).

TYPE	The message type (output). The following values can be returned:
	`AMMT-DATAGRAM`
	`AMMT-REQUEST`
	`AMMT-REPLY`
	`AMMT-REPORT`

COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSREBY (read bytes)

Reads up to the specified number of data bytes from the message object, starting at the current data offset. The data offset must be positioned before the end of the data for the read to be successful (see “AMMSSTDO (set data offset)” on page 333). AMMSREBY will advance the data offset by the number of bytes read, leaving the offset immediately after the last byte read.

```
CALL 'AMMSREBY' USING HMSG, READLEN, DATALEN, DATA, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  READLEN       PIC S9(9) BINARY.
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
READLEN	The maximum number of bytes to be read (input). The data buffer specified by `DATA` must be at least this size. The number of bytes returned is the minimum of `READLEN` and the number of bytes between the data offset and the end of the data.
DATALEN	The number of bytes read (output).
DATA	The read data (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSRS (reset)

Resets the message object to its initial state (see page 317).

```
CALL 'AMMSRS' USING HMSG, OPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  OPTIONS       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
OPTIONS	Reserved, must be specified as zero (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSSTCC (set CCSID)

Sets the coded character set identifier of the message.

```
CALL 'AMMSSTCC' USING HMSG, CCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  CCSID         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
CCSID	The coded character set identifier (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSSTCI (set correl ID)

Sets the correlation identifier of the message.

```
CALL 'AMMSSTCI' USING HMSG, CORRELIDLEN, CORRELID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  CORRELIDLEN   PIC S9(9) BINARY.
01  CORRELID      PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
CORRELIDLEN	The length of the correlation identifier, in bytes (input).
CORRELID	The correlation identifier (input). If CORRELIDLEN is set to zero, the message correlation identifier is reset and the CORRELID parameter will be ignored.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSSELC (set element CCSID)

This specifies the character set to be used for subsequent element message data (including topic and filter data) passed to or returned from the application. Existing elements in the message are unmodified (but will be returned in this character set). The default value of element CCSID is the queue manager CCSID.

```
CALL 'AMMSSELC' USING HMSG, ELEMENTCCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  ELEMENTCCSID  PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
ELEMENTCCSID	The element coded character set identifier (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSSTDO (set data offset)

Sets the data offset for reading or writing byte data. If the data offset is greater than the current data length, it is valid to write data into the message at that offset, but an attempt to read data will result in an error. See "AMMSREBY (read bytes)" on page 331 and "AMMSWRBY (write bytes)" on page 336.

```
CALL 'AMMSSTDO' USING HMSG, OFFSET, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  OFFSET        PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
OFFSET	The offset in bytes (input). Set an offset of zero to read or write from the start of the data.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSSTEN (set encoding)

Sets the encoding of the data in the message.

```
CALL 'AMMSSTEN' USING HMSG, ENCODING, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  ENCODING      PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

ENCODING The encoding of the message (input). It can take one of the following values:

```
AMENC-NATIVE
AMENC-NORMAL
AMENC-NORMAL-FLOAT-390
AMENC-REVERSED
AMENC-REVERSED-FLOAT-390
AMENC-UNDEFINED
```

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSSTFO (set format)

Sets the format of the message.

```
CALL 'AMMSSTFO' USING HMSG, FORMATLEN, FORMAT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  FORMATLEN     PIC S9(9) BINARY.
01  FORMAT        PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
FORMATLEN	The length of the format, in bytes (input).

FORMAT	The format of the message (input). It can take one of the following values, or an application defined string:
	`AMFMT-NONE`
	`AMFMT-STRING`
	`AMFMT-RF-HEADER`
	If set to AMFMT-NONE, the default format for the sender will be used (if available).

COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSSTGS (set group status)

Sets the group status of the message. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group. Once you start sending messages in a group, you must complete the group before sending any messages that are not in the group.

If you specify AMGRP-MIDDLE-MSG-IN-GROUP or AMGRP-LAST-MSG-IN-GROUP without specifying AMGRP-FIRST-MSG-IN-GROUP, the behavior is the same as for AMGRP-FIRST-MSG-IN-GROUP and AMGRP-ONLY-MSG-IN-GROUP respectively.

If you specify AMGRP-FIRST-MSG-IN-GROUP out of sequence, the behavior is the same as for AMGRP-MIDDLE-MSG-IN-GROUP.

```
CALL 'AMMSSTGS' USING HMSG, STATUS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  STATUS        PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).

STATUS	The group status (input). It can take one of the following values:
	`AMGRP-MSG-NOT-IN-GROUP`
	`AMGRP-FIRST-MSG-IN-GROUP`
	`AMGRP-MIDDLE-MSG-IN-GROUP`
	`AMGRP-LAST-MSG-IN-GROUP`
	`AMGRP-ONLY-MSG-IN-GROUP`

COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSSTRC (set report code)

Sets the feedback code type for a message. This is meaningful only for a message of type AMMT-REPORT.

```
CALL 'AMMSSTRC' USING HMSG,  REPCODE,  COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  REPCODE       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).

REPCODE	The feedback (or report code) value (input). In addition to application defined values, the following can be set
	`AMFB-NONE`
	`AMFB-CODE-EXPIRATION`
	`AMFB-CODE-COA`
	`AMFB-CODE-COD`

COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSSTTY (set type)

Sets the message type for a message. If a response message is requested with a publish, subscribe, or unsubscribe request, the specified value is ignored and message type AMMT-REQUEST is used. If the value specified is AMMT-DATAGRAM, this is overridden when requesting or sending a response message (by AMMT-REQUEST and AMMT-RESPONSE, respectively).

```
CALL 'AMMSSTTY' USING HMSG, MSGTYPE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  MSGTYPE       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).

MSGTYPE	The message type to be set (input). It should be set to, or replaced by, one of the following values:
	`AMMT-DATAGRAM`
	`AMMT-REQUEST`
	`AMMT-REPLY`
	`AMMT-REPORT`

COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSWRBY (write bytes)

Writes the specified number of data bytes into the message object, starting at the current data offset. See “AMMSSTDO (set data offset)” on page 333.

If the data offset is not at the end of the data, existing data is overwritten. If the data offset is set beyond the current data length, the message data between the data length and the data offset is undefined. This feature enables applications to construct messages in a non-sequential manner, but care must be taken to ensure that a message is completely filled with data before it is sent.

AMMSWRBY will advance the data offset by the number of bytes written, leaving it immediately after the last byte written.

```
  CALL 'AMMSWRBY' USING HMSG, WRITELEN, BYTEDATA, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HMSG          PIC S9(9) BINARY.
01  WRITELEN      PIC S9(9) BINARY.
01  BYTEDATA      PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HMSG	The message handle returned by AMSECRMS (input).
WRITELEN	The number of bytes to be written (input).
BYTEDATA	The data bytes (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

Sender interface functions

A *sender* object encapsulates a WebSphere MQ object descriptor (MQOD) structure. This represents a WebSphere MQ queue on a local or remote queue manager. An open sender service is always associated with an open connection object (such as a queue manager connection). Support is also included for dynamic sender services (those that encapsulate model queues). The required sender service object definitions can be provided from a repository, or created without a repository definition by defaulting to the existing queue objects on the local queue manager.

The high-level functions AMHSNMS (send message), AMHSNRQ (send request), and AMHSNRS (send response) call these interface functions as required to open the sender service and send a message. Additional calls are provided here to give the application program extra functionality.

A sender service object must be created before it can be opened. This is done implicitly using the high-level functions, or the AMSECRSN (create sender) session interface functions.

A *response* sender service is a special type of sender service used for sending a response to a request message. It must be created using the default definition, and not a definition stored in a repository (see “Services, policies, and policy handlers” on page 491). Once created, it must not be opened until used in its correct context as a response sender when receiving a request message with AMRCRC (receive) or AMHRCRQ (receive request). When opened, its queue and queue manager properties are modified to reflect the *ReplyTo* destination specified in the message being received. When first used in this context, the sender service becomes a response sender service.

AMSNCLEC (clear error codes)

Clears the error codes in the sender object.

```
CALL 'AMSNCLEC' USING HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSENDER       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSENDER	The sender handle returned by AMSECRSN (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

COBOL sender interface

AMSNCL (close)

Closes the sender service.

```
CALL 'AMSNCL' USING HSENDER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSENDER       PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSENDER	The sender handle returned by AMSECRSN (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSNGTCC (get CCSID)

Gets the coded character set identifier of the sender service. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the sender must perform CCSID conversion of the message before it is sent.

```
CALL 'AMSNGTCC' USING HSENDER, CCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSENDER       PIC S9(9) BINARY.
01  CCSID         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSENDER	The sender handle returned by AMSECRSN (input).
CCSID	The coded character set identifier (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSNGTEN (get encoding)

Gets the value used to encode numeric data types for the sender service. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the sender must convert the encoding of the message before it is sent.

```
CALL 'AMSNGTEN' USING HSENDER, ENCODING, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSENDER       PIC S9(9) BINARY.
01  ENCODING      PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSENDER	The sender handle returned by AMSECRSN (input).
ENCODING	The encoding (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSNGTLE (get last error)

Gets the information (completion and reason codes) from the last error for the sender object.

```
CALL 'AMSNGTLE' USING HSENDER, BUFFLEN, STRINGLEN, ERRORTEXT,
                      REASON2, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSENDER       PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  STRINGLEN     PIC S9(9) BINARY.
01  ERRORTEXT     PIC X(n).
01  REASON2       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSENDER	The sender handle returned by AMSECRSN (input).
BUFFLEN	Reserved, must be zero (input).
STRINGLEN	Reserved (output).
ERRORTEXT	Reserved (output).
REASON2	A secondary reason code (output). If `REASON` indicates AMRC-TRANSPORT-WARNING or AMRC-TRANSPORT-ERR, `REASON2` gives a WebSphere MQ reason code.
COMPCODE	Completion code (output).
REASON	Reason code (output). A value of AMRC-SERVICE-HANDLE-ERR indicates that the AMSNGTLE function call has itself detected an error and failed.

AMSNGTNA (get name)

Gets the name of the sender service.

```
CALL 'AMSNGTNA' USING HSENDER, BUFFLEN, NAMELEN, NAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSENDER       PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSENDER	The sender handle returned by AMSECRSN (input).
BUFFLEN	The length in bytes of a buffer in which the name is returned (input).
NAMELEN	The length of the name, in bytes (output).
NAME	The name of the sender service (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

COBOL sender interface

AMSNOP (open)

Opens the sender service.

```
CALL 'AMSNOP' USING HSENDER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSENDER       PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSENDER	The sender handle returned by AMSECRSN (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSNSN (send)

Sends a message to the destination specified by the sender service. If the sender service is not open, it will be opened (if this action is specified in the policy options).

The message data can be passed in the message object, or as a separate parameter (this means that the data is not copied into the message object before the message is sent, which might improve performance, especially if the message data is large).

```
CALL 'AMSNSN' USING HSENDER, HPOLICY, HRECEIVER, HRCVMSG, DATALEN, DATA,
                    HSNDMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSENDER       PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  HRECEIVER     PIC S9(9) BINARY.
01  HRCVMSG       PIC S9(9) BINARY.
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  HSNDMSG       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSENDER	The sender handle returned by AMSECRSN (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
HRECEIVER	The handle of the receiver service to which the response to this message should be sent, if the message being sent is a request message (input). Specify as AMH-NULL-HANDLE if no response is required.
HRCVMSG	The handle of a received message that is being responded to, if this is a response message (input). Specify as AMH-NULL-HANDLE if this is not a response message.
DATALEN	The length of the message data, in bytes (input). If specified as zero, any message data will be passed in the message object (`HSNDMSG`).

DATA	The message data, if DATALEN is non-zero (input).
HSNDMSG	The handle of a message object that specifies the properties of the message being sent (input). If DATALEN is zero, it can also contain the message data. If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-SND-MSG-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSNSNFL (send file)

Sends data from a file.

```
CALL 'AMSNSNFL' USING HSENDER, HPOLICY, OPTIONS, DIRNAMELEN,
                      DIRNAME, FILENAMELEN, FILENAME, HSNDMSG,
                      COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSENDER       PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  OPTIONS       PIC S9(9) BINARY.
01  DIRNAMELEN    PIC S9(9) BINARY.
01  DIRNAME       PIC X(n).
01  FILENAMELEN   PIC S9(9) BINARY.
01  FILENAME      PIC X(n).
01  HSNDMSG       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSENDER	The sender handle returned by AMSECRSN (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
OPTIONS	A reserved field that must be specified as zero.
DIRNAMELEN	A reserved field that must be specified as zero (input).
DIRNAME	A reserved field.
FILENAMELEN	The length of the file name in bytes (input).
FILENAME	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with a receive file call (see "AMRCRCFL (receive file)" on page 351 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
HSNDMSG	The handle of a message object that specifies the properties of the message being sent (input). If specified as AMN-NULL-HANDLE, the system default send message (constant: AMN-SND-MSG-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

If, in your application, you have previously used a message object, referenced by either handle or name, to send or receive data (including AMI elements or topics), you will need to explicitly call AMMSRS (reset message) before re-using the object for sending a file. This applies even if you use the system default message object handle (constant: AMSD-SND-MSG-HANDLE).

Receiver interface functions

A *receiver* object encapsulates a WebSphere MQ object descriptor (MQOD) structure. This represents a local WebSphere MQ queue. An open receiver service is always associated with an open connection object, such as a queue manager connection. Support is also included for dynamic receiver services (that encapsulate model queues). The required receiver service object definitions can be provided from a repository or can be created automatically from the set of existing queue objects available on the local queue manager.

There is a definition type associated with each receiver service:

```
AMDT-UNDEFINED
AMDT-TEMP-DYNAMIC
AMDT-DYNAMIC
AMDT-PREDEFINED
```

A receiver service created from a repository definition will be initially of type AMDT-PREDEFINED or AMDT-DYNAMIC. When opened, its definition type might change from AMDT-DYNAMIC to AMDT-TEMP-DYNAMIC according to the properties of its underlying queue object.

A receiver service created with default values (that is, without a repository definition) will have its definition type set to AMDT-UNDEFINED until it is opened. When opened, this will become AMDT-DYNAMIC, AMDT-TEMP-DYNAMIC, or AMDT-PREDEFINED, according to the properties of its underlying queue object.

AMRCBR (browse)

Browses a message. See the *WebSphere MQ Application Programming Guide* for a full description of the browse options.

```
CALL 'AMRCBR' USING HRECEIVER, HPOLICY, OPTIONS, BUFFLEN, DATALEN, DATA
                    HRCVMSG, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  OPTIONS       PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  HRCVMSG       PIC S9(9) BINARY.
01  HSENDER       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
OPTIONS	Options controlling the browse operation (input). Possible values are:

```
AMBRW-NEXT
AMBRW-FIRST
AMBRW-RECEIVE-CURRENT
AMBRW-DEFAULT          (AMBRW-NEXT)
```

	`AMBRW-RECEIVE-CURRENT` is equivalent to AMRCRC for the message under the browse cursor.
BUFFLEN	The length in bytes of a buffer in which the data is returned (input).
DATALEN	The length of the message data, in bytes (output). This can be set to –1 (input).
DATA	The received message data (output).
HRCVMSG	The handle of the message object for the received message (output).
HSENDER	The handle of the response sender service that the response message must be sent to, if this is a request message (output). This sender service must be created without a repository definition (that is, it must not exist before the AMI session is started), and must be used exclusively for sending a response. Its definition type must be AMDT-UNDEFINED (it will be set to AMDT-RESPONSE by this call). Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

You can return data in the message object or in an application buffer.

To return the data in the message object (HRCVMSG), rather than the application message buffer, set `BUFFLEN` to zero and set both `DATA` and `DATALEN` as non_NULL (not –1).

To return data in an application message buffer:
- set `DATA` as the address of the buffer (that is, non_NULL, not –1)
- set `BUFFLEN` to the length of the buffer

If the value of `BUFFLEN` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a non_NULL value for `DATALEN` (that is, not –1).

To return only the data length:
- set `DATA` to NULL (–1)
- set `BUFFLEN` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

AMRCBRSE (browse selection message)

Browses a message identified by specifying the Correlation ID from the selection message as a selection criterion. See the *WebSphere MQ Application Programming Guide* for a full description of the browse options.

```
CALL 'AMRCBRSE' USING HRECEIVER, HPOLICY, OPTIONS, HSELMSG,
                      BUFFLEN, DATALEN, DATA, HRCVMSG,
                      HRESPONSE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  OPTIONS       PIC S9(9) BINARY.
01  HSELMSG       PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  HRCVMSG       PIC S9(9) BINARY.
01  HRESPONSE     PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER The receiver handle returned by AMSECRRC (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

OPTIONS Options controlling the browse operation (input). Possible values are:

```
AMBRW-NEXT
AMBRW-FIRST
AMBRW-RECEIVE-CURRENT
AMBRW-DEFAULT          (AMBRW-NEXT)
```

`AMBRW-RECEIVE-CURRENT` is equivalent to AMRCRC for the message under the browse cursor.

HSELMSG The handle of a selection message object (input). This is used together with the browse options to identify the message to be received (for example, using the Correlation ID). Specify as AMH_NULL_HANDLE to get the next available message. The CCSID, element CCSID, and encoding values from the selection message define the target values for any data conversions. If target conversion values are required without using the Correlation ID for selection, this can be reset (see **AMMSGELC** on page 321) before invoking the **AMRCBRSE** function.

BUFFLEN The length in bytes of a buffer in which the data is returned (input).

DATALEN The length of the message data, in bytes (output). This can be set to −1 (input).

DATA The received message data (output).

HRCVMSG The handle of the message object for the received message (output).

HSENDER The handle of the response sender service that the response message must be sent to, if this is a request message (output). This sender service must be created without a repository definition (that is, it must not exist before the AMI session is started), and must be used exclusively for sending a response. Its definition type must be AMDT-UNDEFINED (it will be set to AMDT-RESPONSE by this call).

Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.

COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

You can return data in the message object or in an application buffer.

To return the data in the message object (HRCVMSG), rather than the application message buffer, set `BUFFLEN` to zero and set both `DATA` and `DATALEN` as non_NULL (not −1).

To return data in an application message buffer:

- set `DATA` as the address of the buffer (that is, non_NULL, not −1)
- set `BUFFLEN` to the length of the buffer

If the value of `BUFFLEN` is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a non_NULL value for `DATALEN` (that is, not −1).

To return only the data length:

- set `DATA` to NULL (−1)
- set `BUFFLEN` to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

AMRCCLEC (clear error codes)

Clears the error codes in the receiver service object.

```
CALL 'AMRCCLEC' USING HRECEIVER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMRCCL (close)

Closes the receiver service.

```
CALL 'AMRCCL' USING HRECEIVER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMRCGTDT (get definition type)

Gets the definition type of the receiver service.

```
CALL 'AMRCGTDT' USING HRECEIVER, TYPE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  TYPE          PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
TYPE	The definition type (output). It can be one of the following: AMDT-UNDEFINED AMDT-TEMP-DYNAMIC AMDT-DYNAMIC AMDT-PREDEFINED Values other than AMDT-UNDEFINED reflect the properties of the underlying queue object.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMRCGTLE (get last error)

Gets the information (completion and reason codes) from the last error for the receiver object.

```
CALL 'AMRCGTLE' USING HRECEIVER, BUFFLEN, STRINGLEN, ERRORTEXT,
                      REASON2, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  STRINGLEN     PIC S9(9) BINARY.
01  ERRORTEXT     PIC X(n).
01  REASON2       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
BUFFLEN	Reserved, must be zero (input).
STRINGLEN	Reserved (output).
ERRORTEXT	Reserved (output).
REASON2	A secondary reason code (output). If `REASON` indicates AMRC-TRANSPORT-WARNING or AMRC-TRANSPORT-ERR, `REASON2` gives a WebSphere MQ reason code.
COMPCODE	Completion code (output).
REASON	Reason code (output). A value of AMRC-SERVICE-HANDLE-ERR indicates that the AMRCGTLE function call has itself detected an error and failed.

AMRCGTNA (get name)

Gets the name of the receiver service.

```
CALL 'AMRCGTNA' USING HRECEIVER, BUFFLEN, NAMELEN, NAME,
                      COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
BUFFLEN	The length in bytes of a buffer in which the name is returned (input).
NAMELEN	The length of the name, in bytes (output).
NAME	The name of the receiver service (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMRCGTQN (get queue name)

Gets the queue name of the receiver service. This is used to determine the queue name of a permanent dynamic receiver service, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. See also AMRCSTQN (set queue name).

```
CALL 'AMRCGTQN' USING HRECEIVER, BUFFLEN, NAMELEN, QUEUENAME,
                      COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  QUEUENAME     PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
BUFFLEN	The length in bytes of a buffer in which the queue name is returned (input).
NAMELEN	The length of the queue name, in bytes (output).
QUEUENAME	The queue name of the receiver service (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMRCOP (open)

Opens the receiver service.

```
CALL 'AMRCOP' USING HRECEIVER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMRCRC (receive)

Receives a message.

```
  CALL 'AMRCRC' USING HRECEIVER, HPOLICY, HSELMSG, BUFFLEN, DATALEN, DATA,
                      HRCVMSG, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  HSELMSG       PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  HRCVMSG       PIC S9(9) BINARY.
01  HSENDER       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
HSELMSG	The handle of a selection message object (input). This is used to identify the message to be received (for example, using the correlation ID). Specify as AMH-NULL-HANDLE to get the next available message with no selection.
BUFFLEN	The length in bytes of a buffer in which the data is returned (input).
DATALEN	The length of the message data, in bytes (output). Can be specified as -1 (input).
DATA	The received message data (output).
HRCVMSG	The handle of the message object for the received message (input). If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-RCV-MSG-HANDLE) is used. The message object is reset implicitly before the receive takes place.
HSENDER	The handle of the response sender service that a response message must be sent to, if this is a request message (input). This sender service must have been created without a repository definition, and used exclusively for sending a response. Its definition type must be AMDT-UNDEFINED (it will be set to AMDT-RESPONSE by this call).
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

You can return data in the message object or in an application buffer.

To return the data in the message object (HRCVMSG), rather than the application message buffer, set `BUFFLEN` to zero and set both `DATA` and `DATALEN` as non_NULL (not −1).

To return data in an application message buffer:

- set `DATA` as the address of the buffer (that is, non_NULL, not −1)
- set `BUFFLEN` to the length of the buffer

If the value of BUFFLEN is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a non_NULL value for DATALEN (that is, not −1).

To return only the data length without removing the message from the queue:
- set DATA to NULL (−1)
- set BUFFLEN to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

To remove the message from the queue and discard it:
- set DATA or DATALEN to a non_NULL value (that is, not −1)
- set BUFFLEN to zero
- ensure that Accept Truncated Message in the policy receive attributes is selected

The message will be discarded with an AMRC_MSG_TRUNCATED warning.

If AMRC_RECEIVE_BUFF_LEN_ERR is returned, the message length value is returned in DATALEN (if it is non_NULL, that is, not −1), even though the completion code is MQCC_FAILED.

Note that if DATA is NULL (−1) and BUFFLEN is not zero, there is always an AMRC_RECEIVE_BUFF_LEN_ERR error.

AMRCRCFL (receive file)

Receives file message data into a file.

```
CALL 'AMRCRCFL' USING HRECEIVER, HPOLICY, OPTIONS, HSELMSG,
                      DIRNAMELEN, DIRNAME, FILENAMELEN,
                      FILENAME, HRCVMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  OPTIONS       PIC S9(9) BINARY.
01  HSELMSG       PIC S9(9) BINARY.
01  DIRNAMELEN    PIC S9(9) BINARY.
01  DIRNAME       PIC X(n).
01  FILENAMELEN   PIC S9(9) BINARY.
01  FILENAME      PIC X(n).
01  HRCVMSG       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
HSELMSG	The handle of a selection message object (input). This is used to identify the message to be received (for example, using the

	correlation ID). Specify as AMH-NULL-HANDLE to get the next available message with no selection. The CCSID, element CCSID, and encoding values from the selection message define the target values for any data conversions. If target conversion values are required without using the Correlation ID for selection, this can be reset (see **AMMSSTCI** on page 332) before invoking the **AMRCRCFL** function.
DIRNAMELEN	Reserved, must be specified as zero (input). .
DIRNAME	Reserved. .
FILENAMELEN	The length of the file name in bytes (input). .
FILENAME	The name of the file into which the transferred data is to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If blank then the AMI will use the name of the originating file (including any directory prefix) exactly as it was supplied on the send file call. Note that the original file name may not be appropriate for use by the receiver, either because a path name included in the file name is not applicable to the receiving system, or because the sending and receiving systems use different file naming conventions.
HRCVMSG	The handle of the message object to use to receive the file. This parameter is updated with the message properties, for example the Message ID. If the message is a file message, `HRCVMSG` receives the message data. If `HRCVMSG` is specified as AMH-NULL-HANDLE, the default message object (constant AMSD-RCV-MSG-HANDLE) is used. The message object is reset implicitly before the receive takes place.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMRCSTQN (set queue name)

Sets the queue name of the receiver service, when this encapsulates a model queue. This can be used to specify the queue name of a recreated permanent dynamic receiver service, in order to receive messages in a session subsequent to the one in which it was created. See also AMRCGTQN (get queue name).

```
  CALL 'AMRCSTQN' USING HRECEIVER, NAMELEN, QUEUENAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HRECEIVER     PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  QUEUENAME     PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
NAMELEN	The length of the queue name, in bytes (input).
QUEUENAME	The queue name of the receiver service (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

Distribution list interface functions

A *distribution list* object encapsulates a list of sender objects.

AMDLCLEC (clear error codes)

Clears the error codes in the distribution list object.

```
CALL 'AMDLCLEC' USING HDISTLIST, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HDISTLIST     PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMDLCL (close)

Closes the distribution list.

```
CALL 'AMDLCL' USING HDISTLIST, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HDISTLIST     PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMDLGTLE (get last error)

Gets the information (completion and reason codes) from the last error in the distribution list object.

```
CALL 'AMDLGTLE' USING HDISTLIST, BUFFLEN, STRINGLEN, ERRORTEXT,
                      REASON2, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HDISTLIST     PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  STRINGLEN     PIC S9(9) BINARY.
01  ERRORTEXT     PIC X(n).
01  REASON2       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).
BUFFLEN	Reserved, must be zero (input).
STRINGLEN	Reserved (output).
ERRORTEXT	Reserved (output).

REASON2	A secondary reason code (output). If `REASON` indicates AMRC-TRANSPORT-WARNING or AMRC-TRANSPORT-ERR, `REASON2` gives a WebSphere MQ reason code.
COMPCODE	Completion code (output).
REASON	Reason code (output). A value of AMRC-SERVICE-HANDLE-ERR indicates that the AMDLGTLE function call has itself detected an error and failed.

AMDLGTNA (get name)

Gets the name of the distribution list object.

```
CALL 'AMDLGTNA' USING HDISTLIST, BUFFLEN, NAMELEN, NAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HDISTLIST     PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).
BUFFLEN	The length in bytes of a buffer in which the name is returned (input).
NAMELEN	The length of the name, in bytes (output).
NAME	The distribution list object name (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMDLGTSC (get sender count)

Gets a count of the number of sender services in the distribution list.

```
CALL 'AMDLGTSC' USING HDISTLIST, COUNT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HDISTLIST     PIC S9(9) BINARY.
01  COUNT         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).
COUNT	The number of sender services (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMDLGTSH (get sender handle)

Returns the handle of a sender service in the distribution list object with the specified index.

```
CALL 'AMDLGTSH' USING HDISTLIST, HANDLEINDEX, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HDISTLIST     PIC S9(9) BINARY.
01  HANDLEINDEX   PIC S9(9) BINARY.
01  HSENDER       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).
HANDLEINDEX	The index of the required sender service in the distribution list (input). Specify an index of zero to return the first sender service in the list. Use AMDLGTSC to get the number of sender services in the distribution list.
HSENDER	The handle of the sender service (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMDLOP (open)

Opens the distribution list object for each of the destinations in the distribution list. The completion and reason codes returned by this function call indicate if the open was unsuccessful, partially successful, or completely successful.

```
CALL 'AMDLOP' USING HDISTLIST, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HDISTLIST     PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMDLSN (send)

Sends a message to each sender in the distribution list.

```
CALL 'AMDLSN' USING HDISTLIST, HPOLICY, HRECEIVER, DATALEN, DATA,
                    HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HDISTLIST     PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  HRECEIVER     PIC S9(9) BINARY.
01  DATALEN       PIC S9(9) BINARY.
01  DATA          PIC X(n).
01  HMSG          PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
HRECEIVER	The handle of the receiver service to which the response to this message should be sent, if the message being sent is a request message (input). Specify as AMH-NULL-HANDLE if no response is required.
DATALEN	The length of the message data in bytes (input).If specified as zero, any message data will be passed in the message object (`HMSG`).
DATA	The message data, if `DATALEN` is non-zero (input).
HMSG	The handle of a message object that specifies the properties of the message being sent (input). If `DATALEN` is zero, the message object can also contain the message data. If `HMSG` is specified as AMH-NULL-HANDLE, the default send message object (constant: AMSD-SND-MSG-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMDLSNFL (send file)

Sends data from a file to each sender in the distribution list.

```
CALL 'AMDLSNFL' USING HDISTLIST, HPOLICY, OPTIONS, DIRNAMELEN,
                    DIRNAME, FILENAMELEN, FILENAME, HMSG,
                    COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HDISTLIST     PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  OPTIONS       PIC S9(9) BINARY.
01  DIRNAMELEN    PIC S9(9) BINARY.
01  DIRNAME       PIC X(n).
01  FILENAMELEN   PIC S9(9) BINARY.
01  FILENAME      PIC X(n).
01  HMSG          PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).

HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
OPTIONS	Reserved, must be specified as zero (input).
DIRNAMELEN	Reserved, must be specified as zero (input).
DIRNAME	Reserved.
FILENAMELEN	The length of the file name in bytes (input).
FILENAME	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with a receive file call (see "AMRCRCFL (receive file)" on page 351 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
HMSG	The handle of the message object to use to send the file (input). This can be used to specify the Correlation ID for example. If specified as ANM_NULL_HANDLE, the default send message object (constant: AMSD_SND_MSG_HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

If, in your application, you have previously used a message object, referenced by either handle or name, to send or receive data (including AMI elements or topics), you will need to explicitly call AMMSRS (reset message) before re-using the object for sending a file. This applies even if you use the system default message object handle (constant: AMSD-SND-MSG-HANDLE).

The system default message object handle is used when you set `HMSG` to AMH-NULL-HANDLE.

Publisher interface functions

A *publisher* object encapsulates a sender object. It provides support for publish messages to a publish/subscribe broker.

AMPBCLEC (clear error codes)

Clears the error codes in the publisher object.

```
CALL 'AMPBCLEC' USING HPUBLISHER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPUBLISHER    PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPUBLISHER	The publisher handle returned by AMSECRPB (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMPBCL (close)

Closes the publisher service.

```
CALL 'AMPBCL' USING HPUBLISHER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPUBLISHER    PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPUBLISHER	The publisher handle returned by AMSECRPB (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMPBGTCC (get CCSID)

Gets the coded character set identifier of the publisher service. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the publisher must perform CCSID conversion of the message before it is sent.

```
CALL 'AMPBGTCC' USING HPUBLISHER, CCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPUBLISHER    PIC S9(9) BINARY.
01  CCSID         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPUBLISHER	The publisher handle returned by AMSECRPB (input).
CCSID	The coded character set identifier (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMPBGTEN (get encoding)

Gets the value used to encode numeric data types for the publisher service. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the publisher must convert the encoding of the message before it is sent.

```
CALL 'AMPBGTEN' USING HPUBLISHER, ENCODING, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPUBLISHER    PIC S9(9) BINARY.
01  ENCODING      PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPUBLISHER	The publisher handle returned by AMSECRPB (input).
ENCODING	The encoding (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMPBGTLE (get last error)

Gets the information (completion and reason codes) from the last error for the publisher object.

```
CALL 'AMPBGTLE' USING HPUBLISHER, BUFFLEN, STRINGLEN, ERRORTEXT,
                      REASON2, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPUBLISHER    PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  STRINGLEN     PIC S9(9) BINARY.
01  ERRORTEXT     PIC X(n).
01  REASON2       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPUBLISHER	The publisher handle returned by AMSECRPB (input).
BUFFLEN	Reserved, must be zero (input).
STRINGLEN	Reserved (output).
ERRORTEXT	Reserved (output).
REASON2	A secondary reason code (output). If `REASON` indicates AMRC-TRANSPORT-WARNING or AMRC-TRANSPORT-ERR, `REASON2` gives a WebSphere MQ reason code.
COMPCODE	Completion code (output).
REASON	Reason code (output). A value of AMRC-SERVICE-HANDLE-ERR indicates that the AMPBGTLE function call has itself detected an error and failed.

AMPBGTNA (get name)

Gets the name of the publisher service.

```
  CALL 'AMPBGTNA' USING HPUBLISHER, BUFFLEN, NAMELEN, NAME,
                        COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPUBLISHER    PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPUBLISHER	The publisher handle returned by AMSECRPB (input).
BUFFLEN	The length in bytes of a buffer in which the name is returned (input).
NAMELEN	The length of the name, in bytes (output).
NAME	The publisher object name (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMPBOP (open)

Opens the publisher service.

```
  CALL 'AMPBOP' USING HPUBLISHER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPUBLISHER    PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPUBLISHER	The publisher handle returned by AMSECRPB (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMPBPB (publish)

Publishes a message using the publisher service.

The message data is passed in the message object. There is no option to pass it as a separate parameter as with AMSNSN (this would not improve performance because the MQRFH header must be added to the message data before publishing it).

```
  CALL 'AMPBPB' USING HPUBLISHER, HPOLICY, HRECEIVER, HPUBMSG,
                      COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPUBLISHER    PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  HRECEIVER     PIC S9(9) BINARY.
01  HPUBMSG       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPUBLISHER	The publisher handle returned by AMSECRPB (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
HRECEIVER	The handle of the receiver service to which the response to this publish request should be sent (input). Specify as AMH-NULL-HANDLE if no response is required. This parameter is mandatory if the policy specifies implicit registration of the publisher.
HPUBMSG	The handle of a message object for the publication message (input). If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-SND-MSG-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Subscriber interface functions

A *subscriber* object encapsulates both a sender object and a receiver object. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

AMSBCLEC (clear error codes)

Clears the error codes in the subscriber object.

```
CALL 'AMSBCLEC' USING HSUBSCRIBER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSBCL (close)

Closes the subscriber service.

```
CALL 'AMSBCL' USING HSUBSCRIBER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSBGTCC (get CCSID)

Gets the coded character set identifier of the subscriber′s sender service. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the subscriber must perform CCSID conversion of the message before it is sent.

```
CALL 'AMSBGTCC' USING HSUBSCRIBER, CCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  CCSID         PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
CCSID	The coded character set identifier (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSBGTDT (get definition type)

Gets the definition type of the subscriber′s receiver service.

```
CALL 'AMSBGTDT' USING HSUBSCRIBER, TYPE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  TYPE          PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).

TYPE	The definition type (output). It can be:
	`AMDT-UNDEFINED`
	`AMDT-TEMP-DYNAMIC`
	`AMDT-DYNAMIC`
	`AMDT-PREDEFINED`

COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSBGTEN (get encoding)

Gets the value used to encode numeric data types for the subscriber's sender service. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the subscriber must convert the encoding of the message before it is sent.

```
CALL 'AMSBGTEN' USING HSUBSCRIBER, ENCODING, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  ENCODING      PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
ENCODING	The encoding (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSBGTLE (get last error)

Gets the information (completion and reason codes) from the last error for the subscriber object.

```
CALL 'AMSBGTLE' USING HSUBSCRIBER, BUFFLEN, STRINGLEN, ERRORTEXT,
                      REASON2, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  STRINGLEN     PIC S9(9) BINARY.
01  ERRORTEXT     PIC X(n).
01  REASON2       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
BUFFLEN	Reserved, must be zero (input).
STRINGLEN	Reserved (output).
ERRORTEXT	Reserved (output).
REASON2	A secondary reason code (output). If `REASON` indicates AMRC-TRANSPORT-WARNING or AMRC-TRANSPORT-ERR, `REASON2` gives a WebSphere MQ reason code.
COMPCODE	Completion code (output).
REASON	Reason code (output). A value of AMRC-SERVICE-HANDLE-ERR indicates that the AMSBGTLE function call has itself detected an error and failed.

AMSBGTNA (get name)

Gets the name of the subscriber object.

```
  CALL 'AMSBGTNA' USING HSUBSCRIBER, BUFFLEN, NAMELEN, NAME,
                        COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
BUFFLEN	The length in bytes of a buffer in which the name is returned (input).
NAMELEN	The length of the name, in bytes (output).
NAME	The subscriber object name (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSBGTQN (get queue name)

Gets the queue name of the subscriber's receiver service object. This can be used to determine the queue name of a permanent dynamic receiver service, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. See also AMSBSTQN (set queue name).

```
  CALL 'AMSBGTQN' USING HSUBSCRIBER, BUFFLEN, STRINGLEN, QUEUENAME,
                        COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  STRINGLEN     PIC S9(9) BINARY.
01  QUEUENAME     PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
BUFFLEN	The length in bytes of a buffer in which the queue name is returned (input).
STRINGLEN	The length of the queue name, in bytes (output).
QUEUENAME	The queue name (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

COBOL subscriber interface

AMSBOP (open)

Opens the subscriber service.

```
  CALL 'AMSBOP' USING HSUBSCRIBER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSBRC (receive)

Receives a message, normally a publication, using the subscriber service. The message data, topic and other elements can be accessed using the message interface functions (see page 317).

The message data is passed in the message object. There is no option to pass it as a separate parameter as with AMRCRC (this would not give any performance improvement because the MQRFH header has to be removed from the message data after receiving it).

```
  CALL 'AMSBRC' USING HSUBSCRIBER, HPOLICY, HSELMSG, HRCVMSG,
                      COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  HSELMSG       PIC S9(9) BINARY.
01  HRCVMSG       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
HSELMSG	The handle of a selection message object (input). This is used to identify the message to be received (for example, using the correlation ID). Specify as AMH-NULL-HANDLE to get the next available message with no selection.
HRCVMSG	The handle of the message object for the received message (input). If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-RCV-MSG-HANDLE) is used. The message object is reset implicitly before the receive takes place.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSBSTQN (set queue name)

Sets the queue name of the subscriber’s receiver object, when this encapsulates a model queue. This can be used to specify the queue name of a recreated permanent dynamic receiver service, in order to receive messages in a session subsequent to the one in which it was created. See also AMSBGTQN (get queue name).

```
CALL 'AMSBSTQN' USING HSUBSCRIBER, NAMELEN, QUEUENAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  QUEUENAME     PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
NAMELEN	The length of the queue name, in bytes (input).
QUEUENAME	The queue name (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSBSB (subscribe)

Sends a subscribe message to a publish/subscribe broker using the subscriber service, to register a subscription. The topic and other elements can be specified using the message interface functions (see page 317) before sending the message.

Publications matching the subscription are sent to the receiver service associated with the subscriber. By default, this has the same name as the subscriber service, with the addition of the suffix ‘.RECEIVER’.

```
CALL 'AMSBSB' USING HSUBSCRIBER, HPOLICY, HRECEIVER, HSUBMSG,
                    COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  HRECEIVER     PIC S9(9) BINARY.
01  HSUBMSG       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
HRECEIVER	The handle of the receiver service to which the response to this subscribe request should be sent (input). Specify as AMH-NULL-HANDLE if no response is required. This is not the service to which publications will be sent by the broker; they are sent to the receiver service associated with the subscriber (see above).
HSUBMSG	The handle of a message object for the subscribe message (input). If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-SND-MSG-HANDLE) is used.

COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSBUN (unsubscribe)

Sends an unsubscribe message to a publish/subscribe broker using the subscriber service, to deregister a subscription. The topic and other elements can be specified using the message interface functions (see page 317) before sending the message.

To deregister all topics, a policy providing this option must be specified (this is not the default policy). Otherwise, to remove a previous subscription the topic information specified must match that specified on the relevant AMSBSB request.

```
  CALL 'AMSBUN' USING HSUBSCRIBER, HPOLICY, HRECEIVER, HUNSUBMSG,
                      COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HSUBSCRIBER   PIC S9(9) BINARY.
01  HPOLICY       PIC S9(9) BINARY.
01  HRECEIVER     PIC S9(9) BINARY.
01  HUNSUBMSG     PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HSUBSCRIBER	The subscriber handle returned by AMSECRSB (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
HRECEIVER	The handle of the receiver service to which the response to this subscribe request should be sent (input). Specify as AMH-NULL-HANDLE if no response is required.
HUNSUBMSG	The handle of a message object for the unsubscribe message (input). If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-SND-MSG-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Policy interface functions

A *policy* object encapsulates the set of options used for each AMI request (open, close, send, receive, publish and so on). Examples are the priority and persistence of the message, and whether the message is included in a unit of work.

AMPOCLEC (clear error codes)

Clears the error codes in the policy object.

```
CALL 'AMPOCLEC' USING HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPOLICY       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPOLICY	The policy handle returned by AMSECRPO (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMPOGTLE (get last error)

Gets the information (completion and reason codes) from the last error for the policy object.

```
CALL 'AMPOGTLE' USING HPOLICY, BUFFLEN, STRINGLEN, ERRORTEXT,
                      REASON2, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPOLICY       PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  STRINGLEN     PIC S9(9) BINARY.
01  ERRORTEXT     PIC X(n).
01  REASON2       PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPOLICY	The policy handle returned by AMSECRPO (input).
BUFFLEN	Reserved, must be zero (input).
STRINGLEN	Reserved (output).
ERRORTEXT	Reserved (output).
REASON2	A secondary reason code (output). If `REASON` indicates AMRC-TRANSPORT-WARNING or AMRC-TRANSPORT-ERR, `REASON2` gives a WebSphere MQ reason code.
COMPCODE	Completion code (output).
REASON	Reason code (output). A value of AMRC-SERVICE-HANDLE-ERR indicates that the AMPOGTLE function call has itself detected an error and failed.

AMPOGTNA (get name)

Returns the name of the policy object.

```
  CALL 'AMPOGTNA' USING HPOLICY, BUFFLEN, NAMELEN, NAME,
                        COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPOLICY       PIC S9(9) BINARY.
01  BUFFLEN       PIC S9(9) BINARY.
01  NAMELEN       PIC S9(9) BINARY.
01  NAME          PIC X(n).
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPOLICY	The policy handle returned by AMSECRPO (input).
BUFFLEN	The length in bytes of a buffer in which the name is returned (input).
NAMELEN	The length of the name, in bytes (output).
NAME	The policy object name (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMPOGTWT (get wait time)

Returns the wait time (in ms) set for this policy.

```
  CALL 'AMPOGTWT' USING HPOLICY, WAITTIME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPOLICY       PIC S9(9) BINARY.
01  WAITTIME      PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPOLICY	The policy handle returned by AMSECRPO (input).
WAITTIME	The wait time, in ms (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMPOSTWT (set wait time)

Sets the wait time for any receive function using this policy.

```
  CALL 'AMPOSTWT' USING HPOLICY, WAITTIME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01  HPOLICY       PIC S9(9) BINARY.
01  WAITTIME      PIC S9(9) BINARY.
01  COMPCODE      PIC S9(9) BINARY.
01  REASON        PIC S9(9) BINARY.
```

HPOLICY	The policy handle returned by AMSECRPO (input).
WAITTIME	The wait time (in ms) to be set in the policy (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

Part 5. The Java interface

Chapter 14. Using the Application Messaging Interface in Java

The Application Messaging Interface for Java (amJava) provides a Java style of programming, while being consistent with the object-style interface of the Application Messaging Interface for C. It uses a Java Native Interface (JNI) library, so it cannot be used to write Applets to run in a browser environment.

This chapter describes the following:
- "Structure of the AMI"
- "Writing applications in Java" on page 379
- "Building Java applications" on page 389

Note that the term *object* is used in this book in the object-oriented programming sense, not in the sense of WebSphere MQ 'objects' such as channels and queues.

Structure of the AMI

The following classes are provided:

Base classes

AmSessionFactory	Creates AmSession objects.
AmSession	Creates objects within the AMI session, and controls transactional support.
AmMessage	Contains the message data, message ID and correlation ID, and options that are used when sending or receiving a message (most of which come from the policy definition).
AmSender	This is a service that represents a destination (such as a WebSphere MQ queue) to which messages are sent.
AmReceiver	This is a service that represents a source (such as an a WebSphere MQ queue) from which messages are received.
AmDistributionList	Contains a list of sender services to provide a list of destinations.
AmPublisher	Contains a sender service where the destination is a publish/subscribe broker.
AmSubscriber	Contains a sender service (to send subscribe and unsubscribe messages to a publish/subscribe broker) and a receiver service (to receive publications from the broker).
AmPolicy	Defines how the message should be handled, including items such as priority, persistence, and whether it is included in a unit of work.

Interface and helper classes

AmObject	This is a Java interface, which is implemented by the base classes listed previously (with the exception of AmSessionFactory).
AmConstants	This encapsulates all of the constants needed by amJava.
AmElement	This encapsulates name/value pairs that can be added to AmMessage objects.
AmStatus	This encapsulates the error status of amJava objects.

Exception classes

AmException	This is the base Exception class for amJava; all other amJava Exceptions inherit from this class.
AmErrorException	An Exception of this type is raised when an amJava object experiences an error with a severity level of FAILED (CompletionCode = AMCC_FAILED).
AmWarningException	An Exception of this type is raised when an amJava object experiences an error with a severity level of WARNING (CompletionCode = AMCC_WARNING), provided that warnings have been enabled using the **enableWarnings** method.

Using the repository

You can run AMI applications with or without a repository. If you don’t have a repository, you can create an object by specifying its name in a method. It will be created using the appropriate system provided definition (see “System provided definitions” on page 492).

If you have a repository, and you specify the name of an object in a method that matches a name in the repository, the object will be created using the repository definition. (If no matching name is found in the repository, the system provided definition will be used.)

System default objects

The set of system default objects created in C is not accessible directly in Java, but the SYSTEM.DEFAULT.POLICY (constant: AMSD_POL) is used to provide default behavior when a policy is not specified. Objects with identical properties to the system default objects can be created for use in Java using the built-in definitions (see “System provided definitions” on page 492).

Writing applications in Java

This section gives a number of examples showing how to access the Application Messaging Interface using Java.

Many of the method calls are overloaded and in some cases this results in default objects being used. One example of this is the AmPolicy object which can be passed on many of the methods. For example:

Method overloading

```
mySender.send(mySendMessage, myPolicy);

mySender.send(mySendMessage);
```

If a policy has been created to provide specific send behavior, use the first example. However, if the default policy is acceptable, use the second example.

The defaulting of behavior using method overloading is used throughout the examples.

Creating and opening objects

Before using the AMI, you must create and open the required objects. Objects are created with names, which might correspond to named objects in the repository. In the case of the creation of a response sender (`myResponder`) in the following example, the default name for a response type object is specified using the **AmConstants** helper class, so the object is created with default responder values.

Creating AMI objects

```
mySessionFactory = new AmSessionFactory("MY.SESSION.FACTORY");
mySession = mySessionFactory.createSession("MY.SESSION");
myPolicy = mySession.createPolicy("MY.POLICY");

mySender = mySession.createSender("AMT.SENDER.QUEUE");
myReceiver = mySession.createReceiver("AMT.RECEIVER.QUEUE");
myResponder = mySession.createSender(AmConstants.AMDEF_RSP_SND);

mySendMessage = mySession.createMessage("MY.SEND.MESSAGE");
myReceiveMessage = mySession.createMessage("MY.RECEIVE.MESSAGE");
```

The objects are then opened. In the following examples, the session object is opened with the default policy, whereas the sender and receiver objects are opened with a specified policy (`myPolicy`).

Opening the AMI objects

```
mySession.open();
mySender.open(myPolicy);
myReceiver.open(myPolicy);
```

Sending messages

The examples in this section show how to send a datagram (send and forget) message. First, the message data is written to the `mySendMessage` object. Data is always sent in byte form, so the Java **getBytes** method is used to extract the String data as bytes before adding to the message.

Writing data to a message object

```
String  dataSent = new String("message to be sent");
mySendMessage.writeBytes(dataSent.getBytes());
```

Next, the message is sent using the sender service `mySender`.

Sending a message

```
mySender.send(mySendMessage);
```

The policy used is either the default policy for the service, if specified, or the system default policy. The message attributes are set from the policy or service, or the default for the messaging transport.

When more control is needed you can pass a policy object:

Sending a message with a specified policy

```
mySender.send(mySendMessage, myPolicy);
```

The policy controls the behavior of the send command. In particular, the policy specifies whether the send is part of a unit of work, the priority, persistence and expiry of the message and whether policy components should be invoked. Whether the queue should be implicitly opened and left open can also be controlled.

To send a message to a distribution list, for instance `myDistList`, use it as the sender service:

Sending a message to a distribution list

```
myDistList.send(mySendMessage);
```

You can set an attribute such as the *`Format`* before the message is sent, to override the default in the policy or service.

Setting an attribute in a message

```
mySendMessage.setFormat(myFormat):
```

Similarly, after a message has been sent you can retrieve an attribute such as the *`MessageID`*.

Getting an attribute from a message

```
msgId = mySendMessage.getMessageId();
```

For details of the message attributes that you can set and get, see "AmMessage" on page 394.

When a message object is used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see **reset** on page 416) and rebuild it each time.

Sample program

For more details, refer to the `SendAndForget.java` sample program (see "Sample programs for iSeries, UNIX, and Windows" on page 480).

Receiving messages

The next example shows how to receive a message from the receiver service `myReceiver`, and to read the data from the message object `myReceiveMessage`.

Receiving a message and retrieving the data

```
myReceiver.receive(myReceiveMessage);
data = myReceiveMessage.readBytes(myReceiveMessage.getDataLength());
```

The policy used will be the default for the service if defined, or the system default policy. Greater control of the behavior of the receive can be achieved by passing a policy object.

Receiving a message with a specified policy

```
myReceiver.receive(myReceiveMessage, myPolicy);
```

The policy can specify the wait interval, whether the call is part of a unit of work, whether the message should be code page converted, whether all the members of a group must be there before any members can be read, and how to deal with backout failures.

To receive a specific message using its correlation ID, create a selection message object and set its *`CorrelId`* attribute to the required value. The selection message is then passed as a parameter on the receive.

Receiving a specific message using the correlation ID

```
mySelectionMessage = mySession.createMessage("MY.SELECTION.MESSAGE");
mySelectionMessage.setCorrelationId(myCorrelId);
myReceiver.receive(myReceiveMessage, mySelectionMessage, myPolicy);
```

As before, the policy is optional.

You can view the attributes of the message just received, such as the *`Encoding`*.

Getting an attribute from the message

```
encoding = myReceiveMessage.getEncoding();
```

Sample program

For more details, refer to the `Receiver.java` sample program (see “Sample programs for iSeries, UNIX, and Windows” on page 480).

Request/response messaging

In the *request/response* style of messaging, a requester (or client) application sends a request message and expects to receive a response message back. The responder (or server) application receives the request message and produces the response message (or messages) which it sends back to the requester application. The responder application uses information in the request message to know how to send the response message back to the requester.

In the following examples ‘my’ refers to the requesting application (the client); ‘your’ refers to the responding application (the server).

The requester sends a message as described in “Sending messages” on page 379, specifying the service (`myReceiver`) to which the response message should be sent.

Sending a request message

```
mySender.send(mySendMessage, myReceiver);
```

A policy object can also be specified if required.

The responder receives the message as described in “Receiving messages” on page 381, using its receiver service (`yourReceiver`). It also receives details of the response service (`yourResponder`) for sending the response.

Receiving the request message

```
yourReceiver.receive(yourReceiveMessage, yourResponder);
```

A policy object can be specified if required, as can a selection message object (see “Receiving messages” on page 381).

The responder sends its response message (`yourReplyMessage`) to the response service, specifying the received message to which this is a response.

Sending a response to the request message

```
yourResponder.send(yourReplyMessage, yourReceiveMessage);
```

Finally, the requester application receives the response (`myResponseMessage`), which is correlated with the original message it sent (`mySendMessage`).

Receiving the response message

```
myReceiver.receive(myResponseMessage, mySendMessage);
```

In a typical application, the responder might be a server operating in a loop, receiving requests and replying to them. In this case, the message objects should be set to their initial state and the data cleared before servicing the next request. This is achieved as follows:

Resetting the message object

```
yourReceiveMessage.reset();
yourResponseMessage.reset();
```

Sample programs

For more details, refer to the `Client.java` and `Server.java` sample programs (see "Sample programs for iSeries, UNIX, and Windows" on page 480).

File transfer

You can perform file transfers using the `AmSender.sendFile` and `AmReceiver.receiveFile` methods.

Sending a file using the sendFile method

```
mySender.sendFile(mySendMessage, myfilename, myPolicy)
```

Receiving a file using the receiveFile method

```
myReceiver.receiveFile(myReceiveMessage, myfileName, myPolicy)
```

For a complete description of file transfer, refer to "File transfer" on page 21.

Publish/subscribe messaging

With *publish/subscribe* messaging a *publisher* application publishes messages to *subscriber* applications using a *broker*. The message published contains application data and one or more *topic* strings that describe the data. A subscribing application subscribes to topics informing the broker which topics it is interested in. When the broker receives a message from a publisher it compares the topics in the messages to the topics in the subscription from subscribing applications. If they match, the broker forwards the message to the subscribing application.

Data on a particular topic is published as shown in the next example.

Publishing a message on a specified topic

```
String publicationTopic = new String("Weather");
String publicationData = new String("The weather is sunny");

myPubMessage.addTopic(publicationTopic);
myPubMessage.writeBytes(publicationData.getBytes());
myPublisher.publish(myPubMessage, myReceiver);
```

`myReceiver` identifies a response service to which the broker will send any response messages. You can also specify a policy object to modify the behavior of the command.

Writing applications in Java

To subscribe to a publish/subscribe broker you need to specify one or more topics.

Subscribing to a broker on specified topics

```
String weather = new String("Weather");
String birds = new String("Birds");

mySubMessage.addTopic(weather);
mySubMessage.addTopic(birds);
mySubscriber.subscribe(mySubMessage, myReceiver);
```

Broker response messages will be sent to `myReceiver`.

To remove a subscription, add the topic or topics to be deleted to the message object, and use:

Removing a subscription

```
mySubscriber.unsubscribe(myUnsubMessage, myReceiver);
```

To receive a publication from a broker, use:

Receiving a publication

```
mySubscriber.receive(myReceiveMessage, myPolicy);
publication = myReceiveMessage.readBytes(
                myReceiveMessage.getDataLength());
```

You can then use the **getTopicCount** and **getTopic** methods to extract the topic or topics from the message object.

Subscribing applications can also exploit content-based publish/subscribe by passing a filter on subscribe and unsubscribe calls (see “Using WebSphere MQ Integrator Version 2” on page 477).

Sample programs

For more details, refer to the `Publisher.java` and `Subscriber.java` sample programs (see “Sample programs for iSeries, UNIX, and Windows” on page 480).

Using AmElement objects

Publish/subscribe brokers (such as WebSphere MQ Publish/Subscribe) respond to messages that contain name/value pairs to define the commands and options to be carried out. The Application Messaging Interface contains some methods which produce these name/value pairs directly (such as **AmSubscriber.subscribe**). For less commonly used commands, the name/value pairs can be added to a message using an AmElement object.

For example, to send a message containing a 'Request Update' command, use the following:

Using an AmElement object to construct a command message

```
AmElement bespokeElement = new AmElement("MQPSCommand", "ReqUpdate");
mySendMessage.addElement(bespokeElement);
```

You must then send the message, using **AmSender.send**, to the sender service specified for your publish/subscribe broker.

If you use streams with WebSphere MQ Publish/Subscribe, you must add the appropriate name/value element explicitly to the message object.

The message element methods can, in fact, be used to add any element to a message before issuing an publish/subscribe request. Such elements (including topics, which are specialized elements) supplement or override those added implicitly by the request, as appropriate to the individual element type.

The use of name/value elements is not restricted to publish/subscribe applications, they can be used in other applications as well.

Error handling

The **getLastErrorStatus** method always reflects the last most severe error experienced by an object. It can be used to return an AmStatus object encapsulating this error state. Once the error state has been handled, **clearErrorCodes** can be called to reset this error state.

AmJava can raise two types of Exception, one to reflect serious errors and the other to reflect warnings. By default, only AmErrorExceptions are raised. AmWarningExceptions can be enabled using the **enableWarnings** method. Because both are types of AmException, a generic catch block can be used to process all amJava Exceptions.

Enabling AmWarningExceptions might have some unexpected side-effects, especially when an AmObject is returning data such as another AmObject. For example, if AmWarningExceptions are enabled for an AmSession object and an AmSender is created that does not exist in the repository, an AmWarningException will be raised to reflect this fact. If this happens, the AmSender object will not be created, because its creation was interrupted by an Exception. However, there might be times during the life of an AmObject when processing AmWarningExceptions is useful.

For example:

```
try
{
    ...
    mySession.enableWarnings(true);
    mySession.open();
    ...
}
catch (AmErrorException errorEx)
{
    AmStatus sessionStatus = mySession.getLastErrorStatus();
    switch (sessionStatus.getReasonCode())
    {
    case AmConstants.AMRC_XXXX:
        ...
    case AmConstants.AMRC_XXXX:
        ...
    }
    mySession.clearErrorCodes();
}
catch (AmWarningException warningEx)
{
    ...
}
```

Because most of the objects implement the AmObject interface, a generic error handling routine can be written. For example:

```
try
{
    ...
    mySession.open();
    ...
    mySender.send(myMessage):
    ...
    mySender.send(myMessage):
    ...
    mySession.commit();
}
catch(AmException amex);
{
    AmStatus status;
    status = amex.getSource().getLastErrorStatus();
    System.out.println("Object in error; name="+ amex.getSource().getName());
    System.out.println("Object in error; RC="+ status.getReasonCode());
    ...
    amex.getSource().clearErrorCodes();
}
```

The catch block works because all objects that throw the AmException in the try block are AmObjects, and so they all have **getName**, **getLastErrorStatus** and **clearErrorCodes** methods.

Transaction support

Messages sent and received by the AMI can, optionally, be part of a transactional unit of work. A message is included in a unit of work based on the setting of the syncpoint attribute specified in the policy used on the call. The scope of the unit of work is the session handle and only one unit of work may be active at any time.

The API calls used to control the transaction depend on the type of transaction that is used.

- WebSphere MQ messages are the only resource used

A transaction is started by the first message sent or received under syncpoint control, as specified in the policy specified for the send or receive. Multiple messages can be included in the same unit of work. The transaction is committed or backed out using the **commit** or **rollback** method.

- WebSphere MQ is used as an XA transaction coordinator

  The transaction must be started explicitly using the **begin** method before the first recoverable resource (such as a relational database) is changed. The transaction is committed or backed out using an **commit** or **rollback** method.
- An external transaction coordinator is used

  The transaction is controlled using the API calls of an external transaction coordinator (such as CICS, Encina or Tuxedo). The AMI calls are not used but the syncpoint attributed must still be specified in the policy used on the call.

Sending group messages

The AMI allows a sequence of related messages to be included in, and sent as, a message group. Group context information is sent with each message to allow the message sequence to be preserved and made available to a receiving application. To include messages in a group, the group status information of the first and subsequent messages in the group must be set as follows:

```
AMGRP_FIRST_MSG_IN_GROUP for the first message
AMGRP_MIDDLE_MSG_IN_GROUP for all messages other than first and last
AMGRP_LAST_MSG_IN_GROUP for the last message
```

The message status is set using the **AmMessage.setGroupStatus** method. For a complete description of group messages, refer to “Sending group messages” on page 27.

Other considerations

Multithreading

If you are using multithreading with the AMI, a session normally remains locked for the duration of a single AMI call. If you use receive with wait, the session remains locked for the duration of the wait, which might be unlimited (that is, until the wait time is exceeded or a message arrives on the queue). If you want another thread to run while a thread is waiting for a message, it must use a separate session.

AMI handles and object references can be used on a different thread from that on which they were first created for operations that do not involve an access to the underlying (WebSphere MQ) message transport. Functions such as initialize, terminate, open, close, send, receive, publish, subscribe, unsubscribe, and receive publication will access the underlying transport restricting these to the thread on which the session was first opened (for example, using **AmSession.open**). An attempt to issue these on a different thread will cause an error to be returned by WebSphere MQ and a transport error (AMRC_TRANSPORT_ERR) will be reported to the application.

Using WebSphere MQ with the AMI

You must not mix WebSphere MQ function calls with AMI calls within the same process.

Field limits

When string and binary properties such as queue name, message format, and correlation ID are set, the maximum length values are determined by WebSphere MQ, the underlying message transport. See the rules for naming WebSphere MQ objects in the *WebSphere MQ Application Programming Guide*.

Building Java applications

This section contains information that will help you write, prepare, and run your Java application programs on the various operating systems supported by the AMI.

AMI package for Java

AMI provides a jar file that contains all the classes comprising the AMI package for Java.

com.ibm.mq.amt	Java package
com.ibm.mq.amt.jar	Java jar file

This jar file is installed under:

```
/QIBM/ProdData/mqm/amt/Java/lib   (iSeries)

/java/lib                         (UNIX)

\java\lib                         (Windows)
```

See “Directory structure” on page 445 (AIX), page 449 (HP-UX), page453 (iSeries), page 458 (Solaris), or page 462 (Windows).

To use this package you must:

- Import the package into your Java application by using the following statement in that application:

  ```
  import com.ibm.mq.amt.*;
  ```
- Make sure that the AMI jar file is in your CLASSPATH environment variable. See “Setting the runtime environment” on page 444 (AIX), page 448 (HP-UX), page 452 (iSeries), page 457 (Solaris), or page 461 (Windows).

  Do this both in the environment in which your Java program is compiled, and in the environment in which it is run.

Running Java programs

This section explains what you have to do to prepare and run your Java programs on the AIX, iSeries, HP-UX, Solaris, Windows 98, Windows NT, Windows Me, and Windows 2000 operating systems.

The AMI interface for Java makes use of JNI (Java Native Interface) and so requires a platform native library to run successfully. This library must be accessible to your runtime environment. See the following books for versions of Java that are supported by the AMI:

- *WebSphere MQ for AIX, V5.3 Quick Beginnings*
- *WebSphere MQ for HP-UX, V5.3 Quick Beginnings*
- *WebSphere MQ for iSeries V5.3 Quick Beginnings*
- *WebSphere MQ for Solaris, V5.3 Quick Beginnings*
- *WebSphere MQ for Windows, V5.3 Quick Beginnings*

AIX
Make sure that the JNI library `libamtJava.so` is accessible to your runtime environment. You can do this as follows:

```
export LIBPATH=$LIBPATH:/usr/mqm/lib:
```

HP-UX
Make sure that the JNI library `libamtJava.sl` is accessible to your runtime environment. You can do this as follows:

```
export SHLIB_PATH=$SHLIB_PATH:/opt/mqm/lib:
```

Building Java applications

iSeries Make sure that the library QMQMAMI is in the library list.

If you use the iSeries QShell interpreter, you must use the `export -s` command, so that AMI can access the required environment variables.

Solaris Make sure that the JNI library `libamtJava.so` is accessible to your runtime environment. You can do this as follows:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/mqm/lib:
```

Windows Make sure that the JNI library `amtJava.dll` is in one of the directories specified in the PATH environment variable for your runtime environment. For example:

```
SET PATH=%PATH%;C:\MQSeries\bin;
```

If you already have WebSphere MQ installed, it is likely that this environment has already been set up for you.

Once the AMI jar file and the JNI library are referenced in your runtime environment you can run your Java application. For example, to run an application called `mine` that exists in a package `com.xxx.com`, perform:

```
java com.xxx.com.mine
```

Chapter 15. Java interface overview

This chapter contains an overview of the structure of the Application Messaging Interface for Java. Use it to find out what functions are available in this interface.

The Java interface provides sets of methods for each of the classes listed below. The methods available for each class are listed in the following pages. Follow the page references to see the reference information for each method.

Base classes

AmSessionFactory	page 392
AmSession	page 393
AmMessage	page 394
AmSender	page 396
AmReceiver	page 397
AmDistributionList	page 398
AmPublisher	page 399
AmSubscriber	page 400
AmPolicy	page 401

Helper classes

AmConstants	page 402
AmElement	page 402
AmObject	page 402
AmStatus	page 402

Exception classes

AmException	page 403
AmErrorException	page 403
AmWarningExcpetion	page 403

AmSessionFactory

The **AmSessionFactory** class is used to create AmSession objects.

Constructor

Constructor for AmSessionFactory.

AmSessionFactory	page 406

Session factory management

Methods to return the name of an AmSessionFactory object, and to control traces.

getFactoryName	page 406
getLocalHost	page 406
getRepository	page 406
getTraceLevel	page 406
getTraceLocation	page 406
setLocalHost	page 406
setRepository	page 407
setTraceLevel	page 407
setTraceLocation	page 407

Create session

Method to create an AmSession object.

createSession	page 406

AmSession

The **AmSession** object creates and manages all other objects, and provides scope for a unit of work.

Session management

Methods to open and close an AmSession object, to return its name, and to control traces.

open	page 410
close	page 408
getName	page 410
getTraceLevel	page 410
getTraceLocation	page 410

Create objects

Methods to create AmMessage, AmSender, AmReceiver, AmDistributionList AmPublisher, AmSubscriber, and AmPolicy objects.

createMessage	page 409
createSender	page 409
createReceiver	page 409
createDistributionList	page 408
createPublisher	page 409
createSubscriber	page 409
createPolicy	page 409

Transactional processing

Methods to begin, commit and rollback a unit of work.

begin	page 408
commit	page 408
rollback	page 410

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 408
enableWarnings	page 410
getLastErrorStatus	page 410

AmMessage

An **AmMessage** object encapsulates a WebSphere MQ message descriptor (MQMD) structure, and it contains the message data if this is not passed as a separate parameter.

Get values

Methods to get the coded character set ID, correlation ID, encoding, format, group status, message ID and name of the message object.

getCCSID	page 413
getCorrelationId	page 413
getEncoding	page 414
getFormat	page 414
getGroupStatus	page 414
getMessageId	page 415
getName	page 415
getReportCode	page 415
getType	page 416

Set values

Methods to set the coded character set ID, correlation ID, encoding, format, group status, feedback code type, and message type of the message object.

setCCSID	page 416
setCorrelationId	page 416
setEncoding	page 417
setFormat	page 417
setGroupStatus	page 417
setReportCode	page 417
setType	page 418

Reset values

Method to reset the message object to the state it had when first created.

reset	page 416

Read and write data

Methods to read or write byte data to or from the message object, to get and set the data offset, and to get the length of the data.

getDataLength	page 413
getDataOffset	page 413
setDataOffset	page 416
readBytes	page 416
writeBytes	page 418

Publish/subscribe filters

Methods to manipulate filters for content-based publish/subscribe.

addFilter	page 412
deleteFilter	page 412
getFilter	page 414
getFilterCount	page 414

Publish/subscribe topics

Methods to manipulate the topics in a publish/subscribe message.

addTopic	page 412
deleteTopic	page 413
getTopic	page 415
getTopicCount	page 415

Publish/subscribe name/value elements

Methods to manipulate the name/value elements in a publish/subscribe message.

addElement	page 411
deleteElement	page 412
getElement	page 413
getElementCount	page 413
deleteNamedElement	page 412
getNamedElement	page 415
getNamedElementCount	page 415

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 412
enableWarnings	page 413
getLastErrorStatus	page 414

AmSender

An **AmSender** object encapsulates a WebSphere MQ object descriptor (MQOD) structure.

Open and close

Methods to open and close the sender service.

open	page 420
close	page 419

Send

Method to send a message.

send	page 420

Send file

Method to send data from a file

sendFile	page 421

Get values

Methods to get the coded character set ID, encoding and name of the sender service.

getCCSID	page 419
getEncoding	page 419
getName	page 420

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 419
enableWarnings	page 419
getLastErrorStatus	page 420

AmReceiver

An **AmReceiver** object encapsulates a WebSphere MQ object descriptor (MQOD) structure.

Open and close

Methods to open and close the receiver service.

open	page 424
close	page 423

Receive and browse

Methods to receive or browse a message.

receive	page 424
browse	page 422

Receive file

Method to receive file message data into a file.

receiveFile	page 425

Get values

Methods to get the definition type, name and queue name of the receiver service.

getDefinitionType	page 423
getName	page 424
getQueueName	page 424

Set value

Method to set the queue name of the receiver service.

setQueueName	page 425

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 423
enableWarnings	page 423
getLastErrorStatus	page 424

AmDistributionList

An **AmDistributionList** object encapsulates a list of AmSender objects.

Open and close

Methods to open and close the distribution list service.

open	page 426
close	page 426

Send

Method to send a message to the distribution list.

send	page 427

Send file

Method to send date from a file to each sender defined in the distribution list.

sendFile	page 427

Get values

Methods to get the name of the distribution list service, a count of the AmSenders in the list, and one of the AmSenders that is contained in the list.

getName	page 426
getSenderCount	page 426
getSender	page 426

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 426
enableWarnings	page 426
getLastErrorStatus	page 426

AmPublisher

An **AmPublisher** object encapsulates a sender service and provides support for publishing messages to a publish/subscribe broker.

Open and close

Methods to open and close the publisher service.

open	page 429
close	page 428

Publish

Method to publish a message.

publish	page 429

Get values

Methods to get the coded character set ID, encoding and name of the publisher service.

getCCSID	page 428
getEncoding	page 428
getName	page 428

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 428
enableWarnings	page 428
getLastErrorStatus	page 428

AmSubscriber

An **AmSubscriber** object encapsulates both a sender service and a receiver service. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

Open and close

Methods to open and close the subscriber service.

open	page 431
close	page 430

Broker messages

Methods to subscribe to a broker, remove a subscription, and receive a publication from the broker.

subscribe	page 432
unsubscribe	page 432
receive	page 431

Get values

Methods to get the coded character set ID, definition type, encoding, name and queue name of the subscriber service.

getCCSID	page 430
getDefinitionType	page 430
getEncoding	page 430
getName	page 431
getQueueName	page 431

Set value

Method to set the queue name of the subscriber service.

setQueueName	page 431

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 430
enableWarnings	page 430
getLastErrorStatus	page 430

AmPolicy

An **AmPolicy** object encapsulates the options used during AMI operations.

Policy management

Methods to return the name of the policy, and to get and set the wait time when receiving a message.

getName	page 433
getWaitTime	page 433
setWaitTime	page 433

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 433
enableWarnings	page 433
getLastErrorStatus	page 433

Helper classes

A Java Interface, and classes that encapsulate constants, name/value elements, and error status.

AmConstants

Provides access to all the AMI constants.

AmConstants	page 434

AmElement

Constructor for AmElement, and methods to return the name, type, value and version of an element, to set the version, and to return a String representation of the element.

AmElement	page 435
getName	page 435
getValue	page 435
getVersion	page 435
setVersion	page 435
toString	page 435

AmObject

A Java Interface containing methods to return the name of the object, to clear the error codes and to return the last error condition.

clearErrorCodes	page 436
getLastErrorStatus	page 436
getName	page 436

AmStatus

Constructor for AmStatus, and methods to return the completion code, reason code, secondary reason code and status text, and to return a String representation of the AmStatus.

AmStatus	page 437
getCompletionCode	page 437
getReasonCode	page 437
getReasonCode2	page 437
toString	page 437

Exception classes

Classes that encapsulate error and warning conditions. AmErrorException and AmWarningException inherit from AmException.

AmException

Methods to return the completion code and reason code from the Exception, the class name, method name and source of the Exception, and to return a String representation of the Exception.

getClassName	page 438
getCompletionCode	page 438
getMethodName	page 438
getReasonCode	page 438
getSource	page 438
toString	page 438

AmErrorException

Methods to return the completion code and reason code from the Exception, the class name, method name and source of the Exception, and to return a String representation of the Exception.

getClassName	page 439
getCompletionCode	page 439
getMethodName	page 439
getReasonCode	page 439
getSource	page 439
toString	page 439

AmWarningException

Methods to return the completion code and reason code from the Exception, the class name, method name and source of the Exception, and to return a String representation of the Exception.

getClassName	page 440
getCompletionCode	page 440
getMethodName	page 440
getReasonCode	page 440
getSource	page 440
toString	page 440

Java interface overview

Chapter 16. Java interface reference

In the following sections the Java interface methods are listed by the class they refer to. Within each section the methods are listed in alphabetical order.

Note that where constants are shown (for example, AMRC_NONE), they can be accessed using the AmConstants class (for example, AmConstants.AMRC_NONE). See page 434.

Base classes

Note that all of the methods in these classes can throw AmWarningException and AmErrorException (see below). However, by default, AmWarningExceptions are not raised.

Helper classes

Exception classes

AmSessionFactory

The **AmSessionFactory** class is used to create AmSession objects.

AmSessionFactory

Constructor for an AmSessionFactory.

```
AmSessionFactory(String name);
```

`name`	The name of the AmSessionFactory. This is the location of the data files used by the AMI (the repository file and the local host file). The name can be a fully qualified directory that includes the path under which the files are located. Otherwise, see “Local host and repository files (iSeries, UNIX, and Windows)” on page 468 for the location of these files.

createSession

Creates an AmSession object.

```
AmSession  createSession(String name);
```

`name`	The name of the AmSession.

getFactoryName

Returns the name of the AmSessionFactory.

```
String  getFactoryName();
```

getLocalHost

Returns the name of the local host file.

```
String  getLocalHost();
```

getRepository

Returns the name of the repository file.

```
String  getRepository();
```

getTraceLevel

Returns the trace level for the AmSessionFactory.

```
int  getTraceLevel();
```

getTraceLocation

Returns the location of the trace for the AmSessionFactory.

```
String  getTraceLocation();
```

setLocalHost

Sets the name of the AMI local host file to be used by any AmSession created from this AmSessionFactory. (Otherwise, the default host file `amthost.xml` is used.)

```
void  setLocalHost(String  fileName);
```

`fileName`	The name of the file used by the AMI as the local host file. This file must be present on the local file system or an error will be produced upon the creation of an AmSession.

setRepository

Sets the name of the AMI repository to be used by any AmSession created from this AmSessionFactory. (Otherwise, the default repository file `amt.xml` is used.)

```
void  setRepository(String  fileName);
```

`fileName`
Either of the following:

- The name of the file used by the AMI as the repository.

  This file must be present on the local file system or an error will be produced upon the creation of an AmSession.
- A reference to the repository information in LDAP URL format, when repository information is obtained from an LDAP directory.

  For details about specifying an LDAP URL, see “Directory search” on page 512.

setTraceLevel

Sets the trace level for the AmSessionFactory.

```
void  setTraceLevel(int level);
```

`level`	The trace level to be set in the AmSessionFactory. Trace levels are 0 through 9, where 0 represents minimal tracing and 9 represents a fully detailed trace.

setTraceLocation

Sets the location of the trace for the AmSessionFactory.

```
void  setTraceLocation(String location);
```

`location`	The location on the local system where trace files will be written. This location must be a directory, and it must exist before the trace is run.

AmSession

An **AmSession** object provides the scope for a unit of work and creates and manages all other objects, including at least one connection object. Each (WebSphere MQ) connection object encapsulates a single WebSphere MQ queue manager connection. The session object definition specifying the required queue manager connection can be provided by a repository policy definition, or by default will name a single local queue manager with no repository. The session, when deleted, is responsible for releasing memory by closing and deleting all other objects that it manages.

begin

Begins a unit of work in this AmSession, allowing an AMI application to take advantage of the resource coordination provided in WebSphere MQ. The unit of work can subsequently be committed by the **commit** method, or backed out by the **rollback** method. This should be used only when AMI is the transaction coordinator. If available, native coordination APIs (for example CICS or Tuxedo) should be used.

begin is overloaded. The `policy` parameter is optional.

```
void  begin(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

clearErrorCodes

Clears the error codes in the AmSession.

```
void  clearErrorCodes();
```

close

Closes the AmSession, and all open objects owned by it. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

commit

Commits a unit of work that was started by **AmSession.begin**. **commit** is overloaded: the `policy` parameter is optional.

```
void  commit(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

createDistributionList

Creates an AmDistributionList object.

```
AmDistributionList  createDistributionList(String name);
```

`name`	The name of the AmDistributionList. This must match the name of a distribution list defined in the repository.

createMessage

Creates an AmMessage object.

```
AmMessage  createMessage(String name);
```

name	The name of the AmMessage. This can be any name that is meaningful to the application.

createPolicy

Creates an AmPolicy object.

```
AmPolicy  createPolicy(String name);
```

name	The name of the AmPolicy. If it matches a policy defined in the repository, the policy will be created using the repository definition, otherwise it will be created with default values.

createPublisher

Creates an AmPublisher object.

```
AmPublisher  createPublisher(String name);
```

name	The name of the AmPublisher. If it matches a publisher defined in the repository, the publisher will be created using the repository definition, otherwise it will be created with default values (that is, with an AmSender name that matches the publisher name).

createReceiver

Creates an AmReceiver object.

```
AmReceiver  createReceiver(String name);
```

name	The name of the AmReceiver. If it matches a receiver defined in the repository, the receiver will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the receiver name).

createSender

Creates an AmSender object.

```
AmSender  createSender(String name);
```

name	The name of the AmSender. If it matches a sender defined in the repository, the sender will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the sender name).

createSubscriber

Creates an AmSubscriber object.

```
AmSubscriber  createSubscriber(String name);
```

name	The name of the AmSubscriber. If it matches a subscriber defined in the repository, the subscriber will be created using the repository definition, otherwise it will be created with default values (that is, with an AmSender name that matches the subscriber name, and an AmReceiver name that is the same with the addition of the suffix '.RECEIVER').

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(boolean warningsOn);
```

`warningsOn`	If set to `true`, AmWarningExceptions will be raised for this object.

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmSession.

```
String  getName();
```

getTraceLevel

Returns the trace level of the AmSession.

```
int  getTraceLevel();
```

getTraceLocation

Returns the location of the trace for the AmSession.

```
String  getTraceLocation();
```

open

Opens an AmSession using the specified policy. The application profile group of this policy provides the connection definitions enabling the connection objects to be created. The specified library is loaded for each connection and its dispatch table initialized. If the transport type is WebSphere MQ and the WebSphere MQ local queue manager library cannot be loaded, the WebSphere MQ client queue manager is loaded. Each connection object is then opened.

open is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

rollback

Rolls back a unit of work that was started by **AmSession.begin**, or under policy control. **rollback** is overloaded: the `policy` parameter is optional.

```
void  rollback(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmMessage

An **AmMessage** object encapsulates the WebSphere MQ MQMD message properties, and name/value elements such as the topics for publish/subscribe messages. In addition it contains the application data.

The initial state of the message object is:

`CCSID`	default queue manager CCSID
`correlationId`	all zeros
`dataLength`	zero
`dataOffset`	zero
`elementCount`	zero
`encoding`	AMENC_NATIVE
`format`	AMFMT_STRING
`groupStatus`	AMGRP_MSG_NOT_IN_GROUP
`reportCode`	AMFB_NONE
`topicCount`	zero
`type`	AMMT_DATAGRAM

When a message object is used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see **reset** on page 416) and rebuild it each time.

Note that the following methods are only valid after a session has been opened with **AmSession.open**:

addElement	page 411
deleteElement	page 412
getElement	page 413
getElementCount	page 413
deleteNamedElement	page 412
getNamedElement	page 415
getNamedElementCount	page 415
addTopic	page 412
deleteTopic	page 413
getTopic	page 415
getTopicCount	page 415

addElement

Adds a name/value element to an AmMessage object. **addElement** is overloaded: the `element` parameter is required, but the `options` parameter is optional.

```
void  addElement(
   AmElement  element,
   int        options);
```

`element`	The element to be added to the AmMessage.
`options`	The options to be used. This parameter is reserved and must be set to zero.

addFilter

Adds a publish/subscribe filter to an AmMessage object.

```
void  addFilter(String filter);
```

`filter`	The filter to be added to the AmMessage.

addTopic

Adds a publish/subscribe topic to an AmMessage object.

```
void  addTopic(String topicName);
```

`topicName`	The name of the topic to be added to the AmMessage.

clearErrorCodes

Clears the error in the AmMessage object.

```
void  clearErrorCodes();
```

deleteElement

Deletes the element in the AmMessage object at the specified index. Indexing is within all elements of a message, and might include topics (which are specialized elements).

```
void  deleteElement(int index);
```

`index`	The index of the element to be deleted, starting from zero. On completion, elements with higher `index` values than that specified will have those values reduced by one. **getElementCount** gets the number of elements in the message.

deleteFilter

Deletes a publish/subscribe filter in an AmMessage object at the specified index. Indexing is within all filters in the message.

```
void  deleteFilter(int filterIndex);
```

`filterIndex`	The index of the filter to be deleted, starting from zero. **getFilterCount** gets the number of filters in a message.

deleteNamedElement

Deletes the element with the specified name in the AmMessage object, at the specified index. Indexing is within all elements that share the same name.

```
void  deleteNamedElement(
  String  name,
  int     index);
```

`name`	The name of the element to be deleted.
`index`	The index of the element to be deleted, starting from zero. On completion, elements with higher `index` values than that specified will have those values reduced by one. **getNamedElementCount** gets the number of elements in the message with the specified name.

deleteTopic

Deletes a publish/subscribe topic in an AmMessage object at the specified index. Indexing is within all topics in the message.

```
void  deleteTopic(int index);
```

index	The index of the topic to be deleted, starting from zero. **getTopicCount** gets the number of topics in the message.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(boolean warningsOn);
```

warningsOn	If set to `true`, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier used by AmMessage.

```
int  getCCSID();
```

getCorrelationId

Returns the correlation identifier for the AmMessage.

```
byte[]  getCorrelationId();
```

getDataLength

Returns the length of the message data in the AmMessage.

```
int  getDataLength();
```

getDataOffset

Returns the current offset in the message data for reading or writing data bytes.

```
int  getDataOffset();
```

getElement

Returns an element in an AmMessage object at the specified index. Indexing is within all elements in the message, and might include topics (which are specialized elements).

```
AmElement  getElement(int index);
```

index	The index of the element to be returned, starting from zero. **getElementCount** gets the number of elements in the message.

getElementCount

Returns the total number of elements in an AmMessage object. This might include topics (which are specialized elements).

```
int  getElementCount();
```

getEncoding

Returns the value used to encode numeric data types for the AmMessage.

```
int  getEncoding();
```

The following values can be returned:

```
AMENC_NORMAL
AMENC_NORMAL_FLOAT_390
AMENC_REVERSED
AMENC_REVERSED_FLOAT_390
AMENC_UNDEFINED
```

getFilter

Returns the publish/subscribe filter in the AmMessage object at the specified index. Indexing is within all filters.

```
AmString  getFilter(int filterIndex);
```

filterIndex	The index of the filter to be returned, starting from zero. **getElementCount** gets the number of filters in a message.

getFilterCount

Returns the total number of publish/subscribe filters in the AmMessage object.

```
int getFilterCount();
```

getFormat

Returns the format of the AmMessage.

```
String  getFormat();
```

The following values can be returned:

```
AMFMT_NONE
AMFMT_STRING
AMFMT_RF_HEADER
```

getGroupStatus

Returns the group status value for the AmMessage. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group.

```
int  getGroupStatus();
```

The following values can be returned:

```
AMGRP_MSG_NOT_IN_GROUP
AMGRP_FIRST_MSG_IN_GROUP
AMGRP_MIDDLE_MSG_IN_GROUP
AMGRP_LAST_MSG_IN_GROUP
AMGRP_ONLY_MSG_IN_GROUP
```

Alternatively, bitwise tests can be performed using the constants:

```
AMGF_IN_GROUP
AMGF_FIRST
AMGF_LAST
```

getLastErrorStatus

Returns the AmStatus of the last error condition for this object.

```
AmStatus  getLastErrorStatus();
```

getMessageId

Returns the message identifier from the AmMessage object.

```
byte[]  getMessageId();
```

getName

Returns the name of the AmMessage object.

```
String  getName();
```

getNamedElement

Returns the element with the specified name in an AmMessage object, at the specified index. Indexing is within all elements that share the same name.

```
AmElement  getNamedElement(
  String  name,
  int     index);
```

name	The name of the element to be returned.
index	The index of the element to be returned, starting from zero.

getNamedElementCount

Returns the total number of elements with the specified name in the AmMessage object.

```
int  getNamedElementCount(String name);
```

name	The name of the elements to be counted.

getReportCode

Returns the feedback code from an AmMessage of type MQMT_REPORT.

```
int  getReportCode();
```

In addition to application defined values, the following values can be returned:

```
AMFB_NONE
AMFB_CODE_EXPIRATION
AMFB_CODE_COA
AMFB_CODE_COD
```

getTopic

Returns the publish/subscribe topic in the AmMessage object, at the specified index. Indexing is within all topics.

```
String  getTopic(int index);
```

index	The index of the topic to be returned, starting from zero. **getTopicCount** gets the number of topics in the message.

getTopicCount

Returns the total number of publish/subscribe topics in the AmMessage object.

```
int  getTopicCount();
```

getType

Returns the message type from the AmMessage.

```
int  getType();
```

The following values can be returned:

```
AMMT_REQUEST
AMMT_REPLY
AMMT_REPORT
AMMT_DATAGRAM
```

readBytes

Populates a byte array with data from the AmMessage, starting at the current data offset (which must be positioned before the end of the data for the read to be successful). Use **setDataOffset** to specify the data offset. **readBytes** will advance the data offset by the number of bytes read, leaving the offset immediately after the last byte read.

```
byte[]  readBytes(int dataLength);
```

`dataLength`	The maximum number of bytes to be read from the message data. The number of bytes returned is the minimum of `dataLength` and the number of bytes between the data offset and the end of the data.

reset

Resets the AmMessage object to its initial state (see page 411).

reset is overloaded: the `options` parameter is optional.

```
void  reset(int options);
```

`options`	A reserved field that must be set to zero.

setCCSID

Sets the coded character set identifier used by the AmMessage object.

```
void  setCCSID(int codedCharSetId);
```

`codedCharSetId`	The CCSID to be set in the AmMessage.

setCorrelationId

Sets the correlation identifier in the AmMessage object.

```
void  setCorrelationId(byte[] correlId);
```

`correlId` The correlation identifier to be set in the AmMessage. The correlation identifier can be reset by specifying this as a zero length byte array. For example:

```
byte[] myByteArray = new byte[0];
myMessage.setCorrelationId(myByteArray);
```

setDataOffset

Sets the data offset for reading or writing byte data.

```
void  setDataOffset(int dataOffset);
```

`dataOffset`	The data offset to be set in the AmMessage. Set an offset of zero to read or write from the start of the data.

setEncoding

Sets the encoding of the data in the AmMessage object.

```
void  setEncoding(int encoding);
```

encoding The encoding to be used in the AmMessage. It can take one of the following values:

```
AMENC_NORMAL
AMENC_NORMAL_FLOAT_390
AMENC_REVERSED
AMENC_REVERSED_FLOAT_390
AMENC_UNDEFINED
```

setFormat

Sets the format for the AmMessage object.

```
void  setFormat(String format);
```

format The format to be used in the AmMessage. It can take one of the following values:

```
AMFMT_NONE
AMFMT_STRING
AMFMT_RF_HEADER
```

If set to AMFMT_NONE, the default format for the sender will be used (if available).

setGroupStatus

Sets the group status value for the AmMessage. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group. Once you start sending messages in a group, you must complete the group before sending any messages that are not in the group.

If you specify AMGRP_MIDDLE_MSG_IN_GROUP or AMGRP_LAST_MSG_IN_GROUP without specifying AMGRP_FIRST_MSG_IN_GROUP, the behavior is the same as for AMGRP_FIRST_MSG_IN_GROUP and AMGRP_ONLY_MSG_IN_GROUP.

If you specify AMGRP_FIRST_MSG_IN_GROUP out of sequence, the behavior is the same as for AMGRP_MIDDLE_MSG_IN_GROUP.

```
void  setGroupStatus(int groupStatus);
```

groupStatus The group status to be set in the AmMessage. It can take one of the following values:

```
AMGRP_MSG_NOT_IN_GROUP
AMGRP_FIRST_MSG_IN_GROUP
AMGRP_MIDDLE_MSG_IN_GROUP
AMGRP_LAST_MSG_IN_GROUP
AMGRP_ONLY_MSG_IN_GROUP
```

setReportCode

Sets the feedback code used by the AmMessage object. This is meaningful only for a message of type AMMT_REPORT.

```
void setReportCode(int reportCode);
```

reportCode The feedback (or report code) value set in the AmMessage.

| In addition to application defined values, the following values can
| be set:

```
AMFB_NONE
AMFB_CODE_EXPIRATION
AMFB_CODE_COA
AMFB_CODE_COD
```

setType

Sets the message type used by the AmMessage object. If a response message is requested with a publish, subscribe or unsubscribe request, the specified value is ignored and message type AMMT_REQUEST is used. If the value specified is AMMT_DATAGRAM, this is overridden when requesting or sending a response message (by AMMT_REQUEST and AMMT_RESPONSE, respectively).

```
void setType(int type);
```

type	The message type to be set in the AmMessage. It can take one of the following values: AMMT_DATAGRAM AMMT_REQUEST AMMT_REPLY AMMT_REPORT

writeBytes

Writes a byte array into the AmMessage object, starting at the current data offset. If the data offset is not at the end of the data, existing data is overwritten. Use **setDataOffset** to specify the data offset. **writeBytes** will advance the data offset by the number of bytes written, leaving it immediately after the last byte written.

```
void  writeBytes(byte[] data);
```

data	The data to be written to the AmMessage.

AmSender

An **AmSender** object encapsulates a WebSphere MQ object descriptor (MQOD) structure. This represents a WebSphere MQ queue on a local or remote queue manager. An open sender service is always associated with an open connection object (such as a queue manager connection). Support is also included for dynamic sender services (those that encapsulate model queues). The required sender service object definitions can be provided from a repository, or created without a repository definition by defaulting to the existing queue objects on the local queue manager.

The AmSender object must be created before it can be opened. This is done using **AmSession.createSender**.

A *responder* is a special type of AmSender used for sending a response to a request message. It is not created from a repository definition. Once created, it must not be opened until used in its correct context as a responder receiving a request message with **AmReceiver.receive**. When opened, its queue and queue manager properties are modified to reflect the *`ReplyTo`* destination specified in the message being received. When first used in this context, the sender service becomes a responder sender service.

clearErrorCodes

Clears the error codes in the AmSender.

```
void  clearErrorCodes();
```

close

Closes the AmSender. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(boolean warningsOn);
```

`warningsOn`	If set to `true`, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier for the AmSender. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the sender must perform CCSID conversion of the message before it is sent.

```
int  getCCSID();
```

getEncoding

Returns the value used to encode numeric data types for the AmSender. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the sender must convert the encoding of the message before it is sent.

```
int  getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmSender.

```
String  getName();
```

open

Opens an AmSender service. **open** is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

send

Sends a message to the destination specified by the AmSender. If the AmSender is not open, it will be opened (if this action is specified in the policy options).

send is overloaded: the `sendMessage` parameter is required, but the others are optional. `receivedMessage` and `responseService` are used in request/response messaging, and are mutually exclusive.

```
void  send(
  AmMessage   sendMessage,
  AmReceiver  responseService,
  AmMessage   receivedMessage,
  AmPolicy    policy);
```

sendMessage	The message object that contains the data to be sent.
responseService	The AmReceiver to be used for receiving any response to the sent message. If omitted, no response can be received.
receivedMessage	The previously received message which is used for correlation with the sent message. If omitted, the sent message is not correlated with any received message.
policy	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

sendFile

Sends data from a file. To send data from a file, the sendMessage and fileName parameters are required, but the policy is optional. The file data can be received as normal message data by a target application using AmReceiver.receive, or used to reconstruct the file with AmReceiver.receiveFile.

```
void sendFile(
  AmMessage sendMessage,
  String    filename,
  AmPolicy  policy);
```

sendMessage	The message object to use to send the file. This can be used to specify the Correlation ID for example.
fileName	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with the receive file method (see “receiveFile” on page 425 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
policy	The policy to be used. If omitted, the system default policy (name constant: AMSD_POL) is used.

AmReceiver

An **AmReceiver** object encapsulates a WebSphere MQ object descriptor (MQOD) structure. This represents a WebSphere MQ queue on a local or remote queue manager. An open AmReceiver is always associated with an open connection object, such as a queue manager connection. Support is also included for a dynamic AmReceiver (that encapsulates a model queue). The required AmReceiver object definitions can be provided from a repository or can be created automatically from the set of existing queue objects available on the local queue manager.

There is a definition type associated with each AmReceiver:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

An AmReceiver created from a repository definition will be initially of type AMDT_PREDEFINED or AMDT_DYNAMIC. When opened, its definition type might change from AMDT_DYNAMIC to AMDT_TEMP_DYNAMIC according to the properties of its underlying queue object.

An AmReceiver created with default values (that is, without a repository definition) will have its definition type set to AMDT_UNDEFINED until it is opened. When opened, this will become AMDT_DYNAMIC, AMDT_TEMP_DYNAMIC, or AMDT_PREDEFINED, according to the properties of its underlying queue object.

browse

Browses an AmReceiver service. **browse** is overloaded: the `browseMessage` and `options` parameters are required, but the others are optional.

```
void  browse(
  AmMessage   browseMessage,
  int         options,
  AmSender    responseService,
  AmMessage   selectionMessage,
  AmPolicy    policy);
```

`browseMessage` The message object that receives the browse data.

`options` Options controlling the browse operation. Possible values are:

```
AMBRW_NEXT
AMBRW_FIRST
AMBRW_CURRENT
AMBRW_RECEIVE_CURRENT
AMBRW_DEFAULT        (AMBRW_NEXT)
AMBRW_LOCK_NEXT      (AMBRW_LOCK + AMBRW_NEXT)
AMBRW_LOCK_FIRST     (AMBRW_LOCK + AMBRW_FIRST)
AMBRW_LOCK_CURRENT   (AMBRW_LOCK + AMBRW_CURRENT)
AMBRW_UNLOCK
```

`AMBRW_RECEIVE_CURRENT` is equivalent to **AmReceiver.receive** for the message under the browse cursor.

Note that a locked message is unlocked by another browse or receive, even though it is not for the same message.

`responseService`
The AmSender to be used for sending any response to the browsed message. If omitted, no response can be sent.

Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.

`selectionMessage`
A message object which contains the Correlation ID used to selectively browse a message from the AmReceiver. If omitted, the first available message is browsed. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be reset (see **AmMessage.setCorrelationId** on page 416) before invoking the browse method.

`policy` The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

clearErrorCodes

Clears the error codes in the AmReceiver.

```
void  clearErrorCodes();
```

close

Closes the AmReceiver. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy policy);
```

`policy` The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(boolean warningsOn);
```

`warningsOn` If set to `true`, AmWarningExceptions will be raised for this object.

getDefinitionType

Returns the definition type (service type) for the AmReceiver.

```
int  getDefinitionType();
```

The following values can be returned:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

Values other than AMDT_UNDEFINED reflect the properties of the underlying queue object.

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmReceiver.

```
String  getName();
```

getQueueName

Returns the queue name of the AmReceiver. This is used to determine the queue name of a permanent dynamic AmReceiver, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **setQueueName**.)

```
String  getQueueName();
```

open

Opens an AmReceiver service. **open** is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

receive

Receives a message from the AmReceiver service. **receive** is overloaded: the `receiveMessage` parameter is required, but the others are optional.

```
void  receive(
  AmMessage   receiveMessage,
  AmSender    responseService,
  AmMessage   selectionMessage,
  AmPolicy    policy);
```

`receiveMessage`	The message object that receives the data. The message object is reset implicitly before the receive takes place.
`responseService`	The AmSender to be used for sending any response to the received message. If omitted, no response can be sent.
`selectionMessage`	A message object containing the Correlation ID used to selectively receive a message from the AmReceiver. If omitted, the first available message is received. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be be reset (see **AmMessage.setCorrelationId** on page 416) before invoking the receive method.
`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

receiveFile

Receives file message data into a file. To receive data into a file, the receiveMessage and fileName parameters are required, but the others are optional.

```
 void receiveFile(
   AmMessage receiveMessage,
   String    fileName,
   AmMessage selectionMessage,
   AmPolicy  policy);
```

receiveMessage	The message object used to receive the file. This is updated with the message properties, for example the Message ID. If the message is not from a file, the message object receives the data. The message object is reset implicitly before the receive takes place.
fileName	The name of the file to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If NULL or a null string is specified, the AMI will use the name of the originating file (including any directory prefix), exactly as it was supplied on the send file call. Note that the original file name may not be appropriate for use by the receiver, either because a path name included in the file name is not applicable to the receiving system, or because the sending and receiving systems use different file naming conventions.
selectionMessage	A message object containing the Correlation ID used to selectively receive a message from the AmReceiver. If omitted, the first available message is received. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be be reset (see **AmMessage.setCorrelationId** on page 416) before invoking the receive method.
policy	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

setQueueName

Sets the queue name of the AmReceiver (when this encapsulates a model queue). This is used to specify the queue name of a recreated permanent dynamic AmReceiver, in order to receive messages in a session subsequent to the one in which it was created. (See also **getQueueName**.)

```
  void  setQueueName(String queueName);
```

queueName	The queue name to be set in the AmReceiver.

AmDistributionList

An **AmDistributionList** object encapsulates a list of AmSender objects.

clearErrorCodes

Clears the error codes in the AmDistributionList.

```
void  clearErrorCodes();
```

close

Closes the AmDistributionList. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(boolean warningsOn);
```

`warningsOn`	If set to `true`, AmWarningExceptions will be raised for this object.

getLastErrorStatus

Returns the AmStatus of the last error condition of this object.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmDistributionList object.

```
String  getName();
```

getSender

Returns the AmSender in the AmDistributionList object at the index specified. `AmDistributionList.getSenderCount` gets the number of AmSender services in the distribution list.

```
AmSender  getSender(int index);
```

`index`	The index of the AmSender in the AmDistributionList, starting at zero.

getSenderCount

Returns the number of AmSender services in the AmDistributionList object.

```
int  getSenderCount();
```

open

Opens an AmDistributionList object for each of the destinations in the distribution list. **open** is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

send

Sends a message to each AmSender defined in the AmDistributionList object. **send** is overloaded: the `sendMessage` parameter is required, but the others are optional.

```
void  send(
  AmMessage   sendMessage,
  AmReceiver  responseService,
  AmPolicy    policy);
```

`sendMessage`	The message object containing the data to be sent.
`responseService`	The AmReceiver to be used for receiving any response to the sent message. If omitted, no response can be received.
`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

sendFile

Sends data from a file to each AmSender defined in the AmDistributionList object. The `sendMessage` and `fileName` parameters are required to send data from a file, but the policy is optional. The file data can be received as normal message data by a target application using AmReceiver.receive, or used to reconstruct the file with AmReceiver.receiveFile.

```
void  sendFile(
  AmMessage   sendMessage,
  String      fileName,
  AmPolicy    policy);
```

`sendMessage`	The message object to use to send the file. This can be used to specify the Correlation ID, for example.
`fileName`	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with the receive file method (see “receiveFile” on page 425 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
`policy`	The policy to be used. If omitted, the system default policy (name constant: AMSD_POL) is used.

AmPublisher

An **AmPublisher** object encapsulates an AmSender and provides support for publish requests to a publish/subscribe broker.

clearErrorCodes

Clears the error codes in the AmPublisher.

```
void  clearErrorCodes();
```

close

Closes the AmPublisher. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(boolean warningsOn);
```

`warningsOn`	If set to `true`, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier for the AmPublisher. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the publisher must perform CCSID conversion of the message before it is sent.

```
int  getCCSID();
```

getEncoding

Returns the value used to encode numeric data types for the AmPublisher. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the publisher must convert the encoding of the message before it is sent.

```
int  getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmPublisher.

```
String  getName();
```

open

Opens an AmPublisher service. **open** is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (AMSD_POL) is used.

publish

Publishes a message using the AmPublisher. **publish** is overloaded: the `pubMessage` parameter is required, but the others are optional.

```
void  publish(
  AmMessage   pubMessage,
  AmReceiver  responseService,
  AmPolicy    policy);
```

`pubMessage`	The message object that contains the data to be published.
`responseService`	The AmReceiver to which the response to the publish request should be sent. Omit it if no response is required. This parameter is mandatory if the policy specifies implicit registration of the publisher.
`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmSubscriber

An **AmSubscriber** object encapsulates both an AmSender and an AmReceiver. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

clearErrorCodes

Clears the error codes in the AmSubscriber.

```
void  clearErrorCodes();
```

close

Closes the AmSubscriber. **close** is overloaded: the `policy` parameter is optional.

```
void  close(AmPolicy policy);
```

`policy`	The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(boolean warningsOn);
```

`warningsOn`	If set to `true`, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier for the AmSender in the AmSubscriber. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the subscriber must perform CCSID conversion of the message before it is sent.

```
int  getCCSID();
```

getDefinitionType

Returns the definition type for the AmReceiver in the AmSubscriber.

```
int  getDefinitionType();
```

The following values can be returned:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

getEncoding

Returns the value used to encode numeric data types for the AmSender in the AmSubscriber. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the subscriber must convert the encoding of the message before it is sent.

```
int  getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmSubscriber.

```
String  getName();
```

getQueueName

Returns the queue name used by the AmSubscriber to receive messages. This is used to determine the queue name of a permanent dynamic AmReceiver in the AmSubscriber, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **setQueueName**.)

```
String  getQueueName();
```

open

Opens an AmSubscriber. **open** is overloaded: the `policy` parameter is optional.

```
void  open(AmPolicy policy);
```

`policy` The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

receive

Receives a message, normally a publication, using the AmSubscriber. The message data, topic and other elements can be accessed using the message interface methods (see page 411).

receive is overloaded: the `pubMessage` parameter is required, but the others are optional.

```
void  receive(
  AmMessage   pubMessage,
  AmMessage   selectionMessage,
  AmPolicy    policy);
```

`pubMessage` The message object containing the data that has been published. The message object is reset implicitly before the receive takes place.

`selectionMessage`
A message object containing the correlation ID used to selectively receive a message from the AmSubscriber. If omitted, the first available message is received.

`policy` The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

setQueueName

Sets the queue name in the AmReceiver of the AmSubscriber, when this encapsulates a model queue. This is used to specify the queue name of a recreated permanent dynamic AmReceiver, in order to receive messages in a session subsequent to the one in which it was created. (See also **getQueueName**.)

```
void  setQueueName(String queueName);
```

`queueName`
The queue name to be set.

subscribe

Sends a subscribe message to a publish/subscribe broker using the AmSubscriber, to register a subscription. The topic and other elements can be specified using the message interface methods (see page 411) before sending the message.

Publications matching the subscription are sent to the AmReceiver associated with the AmSubscriber. By default, this has the same name as the AmSubscriber, with the addition of the suffix '.RECEIVER'.

subscribe is overloaded: the `subMessage` parameter is required, but the others are optional.

```
void  subscribe(
  AmMessage   subMessage,
  AmReceiver  responseService,
  AmPolicy    policy);
```

`subMessage` The message object that contains the topic subscription data.

`responseService`
The AmReceiver to which the response to this subscribe request should be sent. Omit it if no response is required.

This is not the AmReceiver to which publications will be sent by the broker; they are sent to the AmReceiver associated with the AmSubscriber (see above).

`policy` The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

unsubscribe

Sends an unsubscribe message to a publish/subscribe broker using the AmSubscriber, to deregister a subscription. The topic and other elements can be specified using the message interface methods (see page 411) before sending the message.

unsubscribe is overloaded: the `unsubMessage` parameter is required, but the others are optional.

```
void  unsubscribe(
  AmMessage   unsubMessage,
  AmReceiver  responseService,
  AmPolicy    policy);
```

`unsubMessage` The message object that contains the topics to which the unsubscribe request applies.

`responseService`
The AmReceiver to which the response to this unsubscribe request should be sent. Omit it if no response is required.

`policy` The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmPolicy

An **AmPolicy** object encapsulates details of how the AMI processes the message (for instance, the priority and persistence of the message, how errors are handled, and whether transactional processing is used).

clearErrorCodes

Clears the error codes in the AmPolicy.

```
void  clearErrorCodes();
```

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void  enableWarnings(boolean warningsOn);
```

`warningsOn`	If set to `true`, AmWarningExceptions will be raised for this object.

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmPolicy object.

```
String  getName();
```

getWaitTime

Returns the wait time (in ms) set for this AmPolicy.

```
int  getWaitTime();
```

setWaitTime

Sets the wait time for any **receive** using this AmPolicy.

```
void  setWaitTime(int waitTime);
```

`waitTime`	The wait time (in ms) to be set in the AmPolicy.

AmConstants

This class provides access to the AMI constants listed in Appendix B, “Constants and structures” on page 561.

For example, to use the constant AMRC_NONE (an AMI reason code), specify AmConstants.AMRC_NONE.

Note: Not all of the constants available in the C and C++ programming interfaces are available in Java, because they are not all appropriate in this language. For instance, AmConstants does not contain AMB_TRUE or AMB_FALSE, because the Java language has its own true and false constants and these are used by the AMI for Java.

AmElement

An **AmElement** object encapsulates a name/value pair which can be added to an AmMessage object.

AmElement

Constructor for an AmElement object.

```
AmElement(String name, String value);
```

`name`	The name of the element.
`value`	The value of the element.

getName

Returns the name of the AmElement.

```
String  getName();
```

getValue

Returns the value of the AmElement.

```
String  getValue();
```

getVersion

Returns the version of the AmElement (the default value is AmConstants.AMELEM_VERSION_1).

```
int  getVersion();
```

setVersion

Sets the version of the AmElement.

```
void  setVersion(int version);
```

`version`	The version of the AmElement that is set. It can take the value AmConstants.AMELEM_VERSION_1 or AmConstants.AMELEM_CURRENT_VERSION.

toString

Returns a String representation of the AmElement.

```
String  toString();
```

AmObject

AmObject is a Java Interface. The following classes implement the AmObject interface:

- AmSession
- AmMessage
- AmSender
- AmReceiver
- AmDistributionList
- AmPublisher
- AmSubscriber
- AmPolicy

This allows application programmers to use generic error handling routines.

clearErrorCodes

Clears the error codes in the AmObject.

```
void  clearErrorCodes();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus  getLastErrorStatus();
```

getName

Returns the name of the AmObject.

```
String  getName();
```

AmStatus

An **AmStatus** object encapsulates the error status of other AmObjects.

AmStatus

Constructor for an AmStatus object.

```
AmStatus();
```

getCompletionCode

Returns the completion code from the AmStatus object.

```
int  getCompletionCode();
```

getReasonCode

Returns the reason code from the AmStatus object.

```
int  getReasonCode();
```

getReasonCode2

Returns the secondary reason code from the AmStatus object. (This code is specific to the underlying transport used by the AMI). For WebSphere MQ, the secondary reason code is a WebSphere MQ reason code of type MQRC_xxx.

```
int  getReasonCode2();
```

toString

Returns a String representation of the internal state of the AmStatus object.

```
String  toString();
```

AmException

AmException is the base Exception class; all other Exceptions inherit from this class.

getClassName

Returns the type of object throwing the Exception.

```
String  getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int  getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
String  getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int  getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject  getSource();
```

toString

Returns a String representation of the Exception.

```
String  toString();
```

AmErrorException

An Exception of type **AmErrorException** is raised when an object experiences an error with a severity level of FAILED (CompletionCode = AMCC_FAILED).

getClassName

Returns the type of object throwing the Exception.

```
String  getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int  getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
String  getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int  getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject  getSource();
```

toString

Returns a String representation of the Exception.

```
String  toString();
```

AmWarningException

An Exception of type **AmWarningException** is raised when an object experiences an error with a severity level of WARNING (CompletionCode = AMCC_WARNING).

getClassName

Returns the type of object throwing the Exception.

```
String  getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int  getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
String  getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int  getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject  getSource();
```

toString

Returns a String representation of the Exception.

```
String  toString();
```

Part 6. Setting up an AMI installation

Chapter 17. Installation and sample programs

The Application Messaging Interface is available for the AIX, HP-UX, iSeries, Solaris, Windows 98, Windows NT, Windows Me, Windows 2000, and z/OS platforms.

On the z/OS platform, the Application Messaging Interface is provided as part of IBM WebSphere MQ for z/OS Version 5.3. On the other platforms, it is provided as a SupportPac™ (number MA0F) which can be downloaded from the following Web site:

`http://www.ibm.com/software/ts/mqseries/txppacs/`

On Windows NT and Windows 2000 platforms, an Administration Tool is provided as part of the SupportPac.

You can run the AMI in a WebSphere MQ server or client environment. The Web site gives details of the prerequisite WebSphere MQ (or MQSeries) software, and the disk space required for the SupportPac. Other prerequisites (operating system level, supported compilers) are given in the following books:

- *WebSphere MQ for AIX, V5.3 Quick Beginnings*
- *WebSphere MQ for HP-UX, V5.3 Quick Beginnings*
- *WebSphere MQ for iSeries V5.3 Quick Beginnings*
- *WebSphere MQ for Solaris, V5.3 Quick Beginnings*
- *WebSphere MQ for Windows, V5.3 Quick Beginnings*
- *WebSphere MQ for z/OS Concepts and Planning Guide*

This chapter contains:

Installation on AIX

The AMI package for AIX comes as a compressed archive file, `ma0f_ax.tar.Z`. Uncompress and restore it as follows:

1. Log in as `root`
2. Store `ma0f_ax.tar.Z` in `/tmp`
3. Execute `uncompress -fv /tmp/ma0f_ax.tar.Z`
4. Execute `tar -xvf /tmp/ma0f_ax.tar`
5. Execute `rm /tmp/ma0f_ax.tar`

This creates the following files:

amt100.tar	A standard tar file containing the AMI files
amtInstall	A script file to aid AMI installation
amtRemove	A script file to aid AMI removal
readme	A file containing any product and information updates that have become available since this documentation was produced

Installation

Installation can be carried out manually, or using the **amtInstall** utility.

Manual installation

Restore the tar file `amt100.tar`. This should be done under the base WebSphere MQ directory `/usr/mqm`, so that the AMI tar file restores to a directory structure consistent with WebSphere MQ. This operation usually requires root access. Existing files will be overwritten.

Using amtInstall

1. Log in as `root`
2. Execute `amtInstall <directory>`

where `<directory>` is the directory containing the `amt100.tar` file.

The **amtInstall** utility will unpack the tar file into the correct location and provide the necessary links for your environment. Existing files will be overwritten.

Note: All files and directories created must be accessible to all AMI users. These files are listed in “Directory structure (AIX)” on page 445.

Removing the AMI

Run the **amtRemove** utility to remove all the files that were created by **amtInstall**.

Setting the runtime environment

Make sure that the location of the AMI runtime binary files is added to your PATH environment variable. For example:

```
export PATH=$PATH:/usr/mqm/lib:
```

Note: The previous step is not needed if you used the **amtInstall** utility.

In addition, for the samples:

```
export PATH=$PATH:/usr/mqm/amt/samp/C/bin:/usr/mqm/amt/samp/Cpp/bin:
```

Java programs

When running Java, there are some additional steps.

The AMI classes must be contained in the CLASSPATH, for example:

```
export CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/com.ibm.mq.amt.jar:
```

In addition, for the samples:

```
export CLASSPATH=$CLASSPATH:/usr/mqm/amt/samp/java/bin
                  /com.ibm.mq.amt.samples.jar:
```

Also, to load the AMI library for Java:

```
export LIBPATH=$LIBPATH:/usr/mqm/lib:
```

Next step

Now go to "Local host and repository files (iSeries, UNIX, and Windows)" on page 468 to continue the installation procedure.

Directory structure (AIX)

The AMI tar file contains:

```
./amt/amtsdfts.tst : WebSphere MQ mqsc command file to create default
  WebSphere MQ objects required by the AMI

./amt/amthost.xml : Sample AMI XML file used as the default host file

./amt/amt.dtd : AMI Document Type Definition file on which the AMI
  repository is based

./amt/ipla : The International Program License Agreement file
./amt/li : The License Information file

./amt/inc
   amtc.h : The C header file for the AMI
   amtcpp.hpp : The C++ header file for the AMI
   amtphlc.h: The C header file for the policy handler interface
   amtphmqc.h: The C header file for the policy handler interface (for WebSphere MQ)
   amtxc.h : The C header file for the extended AMI functions
   oamasami.h: The C header file for the OAMAS AMI subset

./amt/handlers
   libamtsphlr.a : C policy handler sample library
   libamtsphlr_r.a : C policy handler sample threaded library

./amt/ldap/
   amtad.vbs : AMI Active Directory Visual Basic script file
   amtad.ldf : AMI Active Directory schema file
   amtsw.ldif : AMI SecureWay schema file for AIX, iSeries, HP-UX, Solaris,
     and Windows
   amtsw390.ldif : AMI SecureWay schema file for OS/390 TDBM
   amtsw390.at : AMI SecureWay schema file for OS/390 RDBM
   amtsw390.oc : AMI SecureWay schema file for OS/390 RDBM

./bin/amtldup : The AMI LDAP directory update program

./java/lib
   com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java

./lib
   libamt.a : The main AMI library
   libamt_r.a : The main AMI threaded library
   libamtXML310.a : The AMI XML parsing library
   libamtXML310_r.a : The AMI threaded XML parsing library
   libamtCpp.a : The AMI C++ library
   libamtCpp_r.a : The AMI C++ threaded library
   libamtJava.so: The AMI JNI library
   libamtICUUC140.a : The AMI codepage translation library
   libamtICUUC140_r.a : The AMI codepage translation threaded library
   libamtICUDATA.a : The AMI codepage translation data library.
   amtcmqm : Dynamic binding stub for Server library
   amtcmqm_r : Dynamic binding stub for WebSphere MQ Server threaded library
   amtcmqic : Dynamic binding stub for WebSphere MQ Client library
   amtcmqic_r : Dynamic binding stub for WebSphere MQ Client threaded library
   amtldap.a : The AMI LDAP library
   amtldap_r.a : The AMI LDAP threaded library

./amt/samp
   amtsamp.tst : WebSphere MQ mqsc command file to create WebSphere MQ objects
      required by AMI samples
   amt.xml : Sample AMI XML repository for use with the AMI samples


./amt/samp/C
   amtsosnd.c : C source for object-level send and forget sample
   amtsorcv.c : C source for object-level receiver sample
   amtsoclt.c : C source for object-level client sample
   amtsosvr.c : C source for object-level server sample
   amtsopub.c : C source for object-level publisher sample
   amtsosub.c : C source for object-level subscriber sample
   amtsofsn.c : C source for object-level send file sample
   amtsofrc.c : C source for object-level receive file sample
   amtsosgs.c : C source for object-level send group sample
   amtsosgr.c : C source for object-level receive group sample
   amtshsnd.c : C source for high-level send and forget sample
   amtshrcv.c : C source for high-level receiver sample
   amtshclt.c : C source for high-level client sample
   amtshsvr.c : C source for high-level server sample
   amtshpub.c : C source for high-level publisher sample
   amtshsub.c : C source for high-level subscriber sample
   amtshfsn.c : C source for high-level send file sample
   amtshfrc.c : C source for high-level receive file sample

./amt/samp/C/bin
   amtsosnd : C object-level send and forget sample program
   amtsorcv : C object-level receiver sample program
   amtsoclt : C object-level client sample program
   amtsosvr : C object-level server sample program
   amtsopub : C object-level publisher sample program
   amtsosub : C object-level subscriber sample program
   amtsofsn : C object-level send file sample program
   amtsofrc : C object-level receive file sample program
   amtsosgs : C object-level send group sample program
   amtsosgr : C object-level receive group sample program
   amtshsnd : C high-level send and forget sample program
   amtshrcv : C high-level receiver sample program
   amtshclt : C high-level client sample program
   amtshsvr : C high-level server sample program
   amtshpub : C high-level publisher sample program
   amtshsub : C high-level subscriber sample program
   amtshfsn : C high-level send file sample program
   amtshfrc : C high-level receive file sample program

./amt/samp/Cpp
   SendForget.cpp : C++ source for send and forget sample
   Receiver.cpp : C++ source for receiver sample
   Client.cpp : C++ source for client sample
   Server.cpp : C++ source for server sample
   Publisher.cpp : C++ source for publisher sample
   Subscriber.cpp : C++ source for subscriber sample
   RcvFile.cpp : C++ source for receive file sample
   SendFile.cpp : C++ source for send file sample

./amt/samp/Cpp/bin
   SendForget : C++ send and forget sample program
   Receiver : C++ receiver sample program
   Client : C++ client sample program
   Server : C++ server sample program
   Publisher : C++ publisher sample program
   Subscriber : C++ subscriber sample program
   RcvFile : C++ source for receive file sample
   SendFile : C++ source for send file sample

./amt/samp/C/handlers
   amtsphlh.h : C header file for policy handler sample
   amtsphlr.c : C source for policy handler sample
   amtsphlr.exp : C export file for policy handler sample

./amt/samp/java
   SendForget.java : Java source for send and forget sample
   Receiver.java : Java source for receiver sample
   Client.java : Java source for client sample
   Server.java : Java source for server sample
   Publisher.java : Java source for publisher sample
   Subscriber.java : Java source for subscriber sample
   RcvFile.java : Java source for receive file sample
   SendFile.java : Java source for send file sample

./amt/samp/java/bin
   com.ibm.mq.amt.samples.jar : The jar file containing the AMI
     samples class files for Java
```

Installation on HP-UX

The AMI package for HP-UX comes as a compressed archive file, `ma0f_hp.tar.Z`. Uncompress and restore it as follows:

1. Log in as `root`
2. Store `ma0f_hp.tar.Z` in `/tmp`
3. Execute `uncompress -fv /tmp/ma0f_hp.tar.Z`
4. Execute `tar -xvf /tmp/ma0f_hp.tar`
5. Execute `rm /tmp/ma0f_hp.tar`

This creates the following files:

amt100.tar	A standard tar file containing the AMI files
amtInstall	A script file to aid AMI installation
amtRemove	A script file to aid AMI removal
readme	A file containing any product and information updates that have become available since this documentation was produced

Installation

Installation can be carried out manually, or using the **amtInstall** utility.

Manual installation

Restore the tar file `amt100.tar`. Do this under the base WebSphere MQ directory `/opt/mqm`, so that the AMI tar file restores to a directory structure consistent with WebSphere MQ. This operation usually requires root access. Existing files will be overwritten.

Using amtInstall

1. Log in as `root`
2. Execute `amtInstall <directory>`

where `<directory>` is the directory containing the `amt100.tar` file.

The **amtInstall** utility will unpack the tar file into the correct location and provide all the necessary links for your environment. Existing files will be overwritten.

Note: All files and directories created must be accessible to all AMI users. These files are listed in “Directory structure (HP-UX)” on page 449.

Removing the AMI

Run the **amtRemove** utility to remove all the files that were created by **amtInstall**.

Setting the runtime environment

Make sure the location of the AMI runtime binary files is added to your PATH environment variable. For example:

```
export PATH=$PATH:/opt/mqm/lib:
```

Note: The previous step is not needed if you used the **amtInstall** utility.

In addition, for the samples:

```
export PATH=$PATH:/opt/mqm/amt/samp/C/bin:/opt/mqm/amt/samp/Cpp/bin:
```

Java programs

When running Java, there are some additional steps.

The AMI classes must be contained in the CLASSPATH, for example:

```
export CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/com.ibm.mq.amt.jar:
```

In addition, for the samples:

```
export CLASSPATH=$CLASSPATH:/opt/mqm/amt/samp/java/bin
                   /com.ibm.mq.amt.samples.jar:
```

Also, to load the AMI library for Java:

```
export SHLIB_PATH=$SHLIB_PATH:/opt/mqm/lib:
```

Next step

Now go to “Local host and repository files (iSeries, UNIX, and Windows)” on page 468 to continue the installation procedure.

Directory structure (HP-UX)

The AMI tar file contains:

```
 ./amt/amtsdfts.tst : WebSphere MQ mqsc command file to create default WebSphere MQ
                      objects required by the AMI

./amt/amthost.xml : Sample AMI XML file used as the default host file

./amt/amt.dtd : AMI Document Type Definition file on which the AMI
                repository is based

./amt/ipla : The International Program License Agreement file
./amt/li : The License Information file

./amt/inc
   amtc.h : The C header file for the AMI
   amtcpp.hpp : The C++ header file for the AMI
   amtphlc.h: The C header file for the policy handler interface
   amtphmqc.h: The C header file for the policy handler interface (for WebSphere MQ)
   amtxc.h : The C header file for the extended AMI functions
   oamasami.h : The C header file for the OAMAS AMI subset

./amt/handlers
   libamtsphlr_r.sl : C policy handler sample library

./amt/ldap/
   amtad.vbs : AMI Active Directory Visual Basic script file
   amtad.ldf : AMI Active Directory schema file
   amtsw.ldif : AMI SecureWay schema file for AIX, iSeries, HP-UX, Solaris,
                and Windows
   amtsw390.ldif : AMI SecureWay schema file for OS/390 TDBM
   amtsw390.at : AMI SecureWay schema file for OS/390 RDBM
   amtsw390.oc : AMI SecureWay schema file for OS/390 RDBM

./bin/amtldup : The AMI LDAP directory update program

./java/lib
   com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java

./lib
   libamt_r.sl : The main AMI threaded library
   libamtXML310_r.sl : The AMI threaded XML parsing library
   libamtCpp_r.sl : The AMI C++ threaded library
   libamtJava.sl: The AMI JNI library
   libamtICUUC140_r.sl : The AMI codepage translation threaded library
   libamtICUDATA.sl: The AMI codepage translation data library.
   amtcmqm_r : Dynamic binding stub for WebSphere MQ Server threaded library
   amtcmqic_r : Dynamic binding stub for WebSphere MQ Client threaded library
   libamtldap_r.sl : The AMI LDAP threaded library

./amt/samp
   amtsamp.tst : WebSphere MQ mqsc command file to create WebSphere MQ objects
                 required by AMI samples
   amt.xml : Sample AMI XML repository for use with the AMI samples


./amt/samp/C
   amtsosnd.c : C source for object-level send and forget sample
   amtsorcv.c : C source for object-level receiver sample
   amtsoclt.c : C source for object-level client sample
   amtsosvr.c : C source for object-level server sample
   amtsopub.c : C source for object-level publisher sample
   amtsosub.c : C source for object-level subscriber sample
   amtsofsn.c : C source for object-level send file sample
   amtsofrc.c : C source for object-level receive file sample
   amtsosgs.c : C source for object-level send group sample
   amtsosgr.c : C source for object-level receive group sample
   amtshsnd.c : C source for high-level send and forget sample
   amtshrcv.c : C source for high-level receiver sample
   amtshclt.c : C source for high-level client sample
   amtshsvr.c : C source for high-level server sample
   amtshpub.c : C source for high-level publisher sample
   amtshsub.c : C source for high-level subscriber sample
   amtshfsn.c : C source for high-level send file sample
   amtshfrc.c : C source for high-level receive file sample

./amt/samp/C/bin
   amtsosnd : C object-level send and forget sample program
   amtsorcv : C object-level receiver sample program
   amtsoclt : C object-level client sample program
   amtsosvr : C object-level server sample program
   amtsopub : C object-level publisher sample program
   amtsosub : C object-level subscriber sample program
   amtsofsn : C object-level send file sample program
   amtsofrc : C object-level receive file sample program
   amtsosgs : C object-level send group sample program
   amtsosgr : C object-level receive group sample program
   amtshsnd : C high-level send and forget sample program
   amtshrcv : C high-level receiver sample program
   amtshclt : C high-level client sample program
   amtshsvr : C high-level server sample program
   amtshpub : C high-level publisher sample program
   amtshsub : C high-level subscriber sample program
   amtshfsn : C high-level send file sample program
   amtshfrc : C high-level receive file sample program

./amt/samp/Cpp
   SendForget.cpp : C++ source for send and forget sample
   Receiver.cpp : C++ source for receiver sample
   Client.cpp : C++ source for client sample
   Server.cpp : C++ source for server sample
   Publisher.cpp : C++ source for publisher sample
   Subscriber.cpp : C++ source for subscriber sample
   RcvFile.cpp : C++ source for receive file sample
   SendFile.cpp : C++ source for send file sample

./amt/samp/Cpp/bin
   SendForget : C++ send and forget sample program
   Receiver : C++ receiver sample program
   Client : C++ client sample program
   Server : C++ server sample program
   Publisher : C++ publisher sample program
   Subscriber : C++ subscriber sample program
   RcvFile : C++ source for receive file sample
   SendFile : C++ source for send file sample

./amt/samp/C/handlers
   amtsphlh.h : C header file for policy handler sample
   amtsphlr.c : C source for policy handler sample
   amtsphlr.exp :  C export file for policy handler sample

./amt/samp/java
   SendForget.java : Java source for send and forget sample
   Receiver.java : Java source for receiver sample
   Client.java : Java source for client sample
   Server.java : Java source for server sample
   Publisher.java : Java source for publisher sample
   Subscriber.java : Java source for subscriber sample
   RcvFile.java : Java source for receive file sample
   SendFile.java : Java source for send file sample

./amt/samp/java/bin
   com.ibm.mq.amt.samples.jar : The jar file containing the AMI
     samples class files for Java
```

Installation on iSeries

The AMI package for iSeries comes as a compressed zip file `ma0f_400.zip`. Uncompress and restore it as follows:

1. Download `ma0f_400.zip` to a directory on your PC.
2. Uncompress the file using the InfoZip Unzip program.

   The file `ma0f_400.sav` is created.
3. Create a save file called MA0F in a suitable library on the iSeries, for example the library QGPL:

   ```
   CRTSAVF FILE(QGPL/MA0F)
   ```
4. Transfer `ma0f_400.sav` into this save file as a binary image.

   If you use FTP to do this, the put command should be similar to:

   ```
   PUT C:\TEMP\MA0F_400.SAV QGPL/MA0F
   ```
5. Install the WebSphere MQ for iSeries AMI, product Id 5724A23, using RSTLICPGM:

   ```
   RSTLICPGM LICPGM(5724A23) DEV(*SAVF) SAVF(QGPL/MA0F)
   ```

   If the Primary Language Feature ID of the system on which you are you are installing is not 2924, you must specify 2924 for the LNG option:

   ```
   RSTLICPGM LICPGM(5724A23) DEV(*SAVF) LNG(2924) SAVF(QGPL/MA0F)
   ```
6. Delete the save file created in Step 3:

   ```
   DLTF FILE(QGPL/MA0F)
   ```

To remove the AMI package from the iSeries, use DLTLICPGM:

```
DLTLICPGM LICPGM(5724A23)
```

Setting the runtime environment for Java programs

To run Java:

- Ensure that the library QMQMAMI is in the library list.

  To add a library, you can use the ADDLIBLE command.
- Ensure that the AMI classes are contained in the CLASSPATH.

  Use the WRKENVVAR command to determine whether a CLASSPATH exists.

  If a CLASSPATH environment variable does not exist, use the ADDENVVAR command to create one. For example:

  ```
  ADDENVVAR ENVVAR(CLASSPATH) VALUE('/QIBM/ProdData/mqm/amt/
                                    java/lib/com.ibm.mq.amt.jar')
  ```

  If a CLASSPATH environment variable already exists, use the CHGENVVAR command to add the following to it:

  ```
  :/QIBM/ProdData/mqm/amt/java/lib/com.ibm.mq.amt.jar
  ```

  To use the supplied samples, include the following in the CLASSPATH:

  ```
  :/QIBM/ProdData/mqm/amt/samp/java/bin/com.ibm.mq.amt.samples.jar
  ```

  Note: Each CLASSPATH entry must be separated by a colon.

Next step

Now go to “Local host and repository files (iSeries, UNIX, and Windows)” on page 468 to continue the installation procedure.

Directory structure (iSeries)

Installation adds the following files:

Library QMQMAMI

```
AMT         *SRVPGM     : The main AMI library
AMT_R       *SRVPGM     : The main AMI threaded library
AMTCPP      *SRVPGM     : The AMI C++ library
AMTCPP_R    *SRVPGM     : The AMI C++  threaded library
AMTJAVA     *SRVPGM     : The AMI JNI library
AMTXML      *SRVPGM     : The main AMI XML parsing library
AMTMSG      *MSGF       : AMT message file
QAMT0050    *PRDDFN     : Product definition file
QAMT0029    *PRDLOD     : Language Product Load file
QAMT0050    *PRDLOD     : Code Product Load file
H           *FILE       : AMI header files
AMTMQSC     *FILE       : MQSC command files
QXMLMSG     *MSGF       : XML message file
AMTSHCLT    *PGM        : C high-level client sample program
AMTSHFRC    *PGM        : C high-level receive file sample program
AMTSHFSN    *PGM        : C high-level send file sample program
AMTSHPUB    *PGM        : C high-level subscriber sample program
AMTSHRCV    *PGM        : C high-level receiver sample program
AMTSHSND    *PGM        : C high-level send and forget sample program
AMTSHSUB    *PGM        : C high-level subscriber sample program
AMTSHSVR    *PGM        : C high-level server sample program
AMTSOCLT    *PGM        : C object-level client sample program
AMTSOFRC    *PGM        : C object-level receive file sample program
AMTSOFSN    *PGM        : C object-level send file sample program
AMTSOPUB    *PGM        : C object-level publisher sample program
AMTSORCV    *PGM        : C object-level receiver sample program
AMTSOSGR    *PGM        : C object-level receive group sample program
AMTSOSGS    *PGM        : C object-level send group sample program
AMTSOSND    *PGM        : C object-level send and forget sample program
AMTSOSUB    *PGM        : C object-level subscriber sample program
AMTSOSVR    *PGM        : C object-level server sample program
CLIENT      *PGM        : C++ client sample program
PUBLISHER   *PGM        : C++ publisher sample program
RECEIVER    *PGM        : C++ receiver sample program
RCVFILE     *PGM        : C++ receive file sample program
SENDFORGET  *PGM        : C++ send and forget sample program
SENDFILE    *PGM        : C++ send file sample program
SERVER      *PGM        : C++ server sample program
SUBSCRIBER  *PGM        : C++ subscriber sample program
QCSRC       *FILE       : C sample files
QCPPSRC     *FILE       : C++ samples
AMTI0X1C    *PGM        : Installation Exit Program
AMTI0XOC    *PGM        : Installation Exit Program
QSRVSRC     *FILE       : Export file for policy handler sample
AMTLDAP     *SRVPGM     : The AMI LDAP library
AMTLDAP_R   *SRVPGM     : The AMI LDAP threaded library
AMTLDUP     *PGM        : The AMI LDAP directory update program
```

Members of QMQMAMI/H

```
AMTC          : The C header file for AMI
AMTCPP        : The C++ header file for AMI
AMTPHLC       : Policy handler interface definition
AMTPHMQC      : Transport-specific policy handler definitions for
                WebSphere MQ
AMTSPHLH      : The C header file for policy handler sample
AMTXC         : AMI extensions for policy handler callback functions
OAMASAMI      : The C header file for the OAMAS AMI subset
```

Note: The members of the file H are copies of the AMI header files in IFS.

Members of QMQMAMI/AMTMQSC

```
AMTSDFTS      : WebSphere MQ mqsc command file to create default
                WebSphere MQ objects required by the AMI
AMTSAMP       : WebSphere MQ mqsc command file to create WebSphere MQ
                objects required by AMI samples
```

Members of QMQMAMI/QCSRC

```
amtshclt      : C source for high-level client sample program
amtshfrc      : C source for high-level receive file sample program
amtshfsn      : C source for high-level send file sample program
amtshpub      : C source for high-level subscriber sample program
amtshrcv      : C source for high-level receiver sample program
amtshsnd      : C source for high-level send and forget sample program
amtshsub      : C source for high-level subscriber sample program
amtshsvr      : C source for high-level server sample program
amtsoclt      : C source for object-level client sample program
amtsofrc      : C source for object-level receive file sample program
amtsofsn      : C source for object-level send file sample program
amtsopub      : C source for object-level publisher sample program
amtsorcv      : C source for object-level receiver sample program
amtsosgr      : C source for object-level receive group sample program
amtsosgs      : C source for object-level send group sample program
amtsosnd      : C source for object-level send and forget sample program
amtsosub      : C source for object-level subscriber sample program
amtsosvr      : C source for object-level server sample program
amtsphlr      : C source for policy handler sample
```

Note: The members of the file QCSRC are copies of the C source files for sample programs in IFS.

Members of QMQMAMI/QSRVSRC

```
amtsphlr      : Export file for policy handler sample
```

Note: The member of the file QSRVSRC is a copy of the export file for the policy handler sample in IFS.

Members of QMQMAMI/QCPPSRC

```
Client        : C++ source for client sample
Publisher     : C++ source for publisher sample
Receiver      : C++ source for receiver sample
RcvFile       : C++ source for receive file sample
SendForget    : C++ source for send and forget sample
SendFile      : C++ source for send file sample
Server        : C++ source for server sample
Subscriber    : C++ source for subscriber sample
```

Note: The members of the file QCPPSRC are copies of the C++ source files for sample programs in IFS.

Library QMQMAMIPHL

```
AMTSPHLR    *SRVPGM     : Sample policy handler library
AMTSPHLR_R  *SRVPGM     : Sample policy handler threaded library
```

IFS

```
/QIBM/ProdData/mqm/amt
    amt.dtd : Document Type Definition file on which the AMI repository
              is based
    amthost.xml  : Sample AMI XML file used as the default host file
    amtsdfts.tst : WebSphere MQ mqsc command file to create default
                   WebSphere MQ objects required by the AMI
    ipla        : The International Program License Agreement file
    li          : The License Information file
    readme      : Product and information updates that became available
```

```
                  after this documentation was produced

 /QIBM/ProdData/mqm/amt/inc
     amtc.h      : The C header file for AMI
     amtcpp.hpp  : The C++ header file for AMI
     amtphlc.h   : The C header file for the policy handler interface
     amtphmqc.h  : The C header file for the policy handler interface
                   (for WebSphere MQ)
     amtxc.h     : The C header file for the extended AMI functions
     oamasami.h  : The C header file for the OAMAS AMI subset


 /QIBM/ProdData/mqm/amt/samp
     amtsamp.tst : WebSphere MQ mqsc command file to create WebSphere MQ
                   objects required by AMI samples
     amt.xml     : Sample AMI XML repository for use with the AMI samples

 /QIBM/ProdData/mqm/amt/samp/C
     amtshclt.c  : C source for high-level client sample program
     amtshfrc.c  : C source for high-level receive file sample program
     amtshfsn.c  : C source for high-level send file sample program
     amtshpub.c  : C source for high-level subscriber sample program
     amtshrcv.c  : C source for high-level receiver sample program
     amtshsnd.c  : C source for high-level send and forget sample program
     amtshsub.c  : C source for high-level subscriber sample program
     amtshsvr.c  : C source for high-level server sample program
     amtsoclt.c  : C source for object-level client sample program
     amtsofrc.c  : C source for object-level receive file sample program
     amtsofsn.c  : C source for object-level send file sample program
     amtsopub.c  : C source for object-level publisher sample program
     amtsorcv.c  : C source for object-level receiver sample program
     amtsosgr.c  : C source for object-level receive group sample program
     amtsosgs.c  : C source for object-level send group sample program
     amtsosnd.c  : C source for object-level send and forget sample program
     amtsosub.c  : C source for object-level subscriber sample program
     amtsosvr.c  : C source for object-level server sample program

/QIBM/ProdData/mqm/amt/samp/C/handlers
     amtsphlr.c         : C source for policy handler sample
     amtsphlr.h         : C include for policy handler sample
     amtsphlr.exp       : Export file for the sample policy handler


 /QIBM/ProdData/mqm/amt/samp/Cpp
     Client.cpp         : C++ source for client sample
     Publisher.cpp      : C++ source for publisher sample
     Receiver.cpp       : C++ source for receiver sample
     RcvFile.cpp        : C++ source for receive file sample
     SendForget.cpp     : C++ source for send and forget sample
     SendFile.cpp       : C++ source for send file sample
     Server.cpp         : C++ source for server sample
     Subscriber.cpp     : C++ source for subscriber sample

/QIBM/ProdData/mqm/amt/samp/java
     Client.java        : Java source for client sample
     Publisher.java     : Java source for publisher sample
     Receiver.java      : Java source for receiver sample
     RcvFile.java       : Java source for receive file sample
     SendForget.java    : Java source for send and forget sample
     SendFile.java      : Java source for send file sample
     Server.java        : Java source for server sample
     Subscriber.java    : Java source for subscriber sample

 /QIBM/ProdData/mqm/amt/samp/java/bin
     com.ibm.mq.amt.samples.jar : The jar file containing the AMI
                                  samples class files for Java
```

Installation on iSeries

```
/QIBM/ProdData/mqm/amt/java/lib
    com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java

/QIBM/ProdData/mqm/amt/ldap
    amtad.vbs       :AMI Active Directory Visual Basic script file
    amtad.ldf       :AMI Active Directory schema file
    amtsw.ldif      :AMI SecureWay schema file for
                     AIX, iSeries, HP-UX, Sun Solaris and Windows
    amtsw390.ldif   :AMI SecureWay schema file for OS/390 TDBM
    amtsw390.at     :AMI SecureWay schema file for OS/390 RDBM
    amtsw390.oc     :AMI SecureWay schema file for OS/390 RDBM

/QIBM/UserData/mqm/amt
    amt.dtd         : Document Type Definition file on which the AMI
                      repository is based
    amthost.xml     : Sample AMI XML file used as the default host file
```

Installation on Solaris

The AMI package for Solaris comes as a compressed archive file, `ma0f_sol.tar.Z`. Uncompress and restore it as follows:

1. Log in as `root`
2. Store `ma0f_sol.tar.Z` in `/tmp`
3. Execute `uncompress -fv /tmp/ma0f_sol.tar.Z`
4. Execute `tar -xvf /tmp/ma0f_sol.tar`
5. Execute `rm /tmp/ma0f_sol.tar`

This creates the following files:

amt100.tar	A standard tar file containing the AMI files
amtInstall	A script file to aid AMI installation
amtRemove	A script file to aid AMI removal
readme	A file containing any product and information updates that have become available since this documentation was produced

Installation

Installation can be carried out manually, or using the **amtInstall** utility.

Manual installation

Restore the tar file `amt100.tar`. This should be done under the base WebSphere MQ directory `/opt/mqm`, so that the AMI tar file restores to a directory structure consistent with WebSphere MQ. This operation usually requires root access. Existing files will be overwritten.

Using amtInstall

1. Log in as `root`
2. Execute `amtInstall <directory>`

where `<directory>` is the directory containing the `amt100.tar` file.

The **amtInstall** utility will unpack the tar file into the correct location and provide the necessary links for your environment. Existing files will be overwritten.

Note: All files and directories created must be accessible to all AMI users. These files are listed in “Directory structure (Solaris)” on page 458.

Removing the AMI

Run the **amtRemove** utility to remove all the files that were created by **amtInstall**.

Setting the runtime environment

Make sure that the location of the AMI runtime binary files is added to your PATH environment variable. For example:

```
export PATH=$PATH:/opt/mqm/lib:
```

Note: The previous step is not needed if you used the **amtInstall** utility.

In addition, for the samples:

```
export PATH=$PATH:/opt/mqm/amt/samp/C/bin:/opt/mqm/amt/samp/Cpp/bin:
```

Java programs

When running Java, there are some additional steps.

The AMI classes must be contained in the CLASSPATH, for example:

```
export CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/com.ibm.mq.amt.jar:
```

In addition, for the samples:

```
export CLASSPATH=$CLASSPATH:/opt/mqm/amt/samp/java/bin
                  /com.ibm.mq.amt.samples.jar:
```

Also, to load the AMI library for Java:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/mqm/lib:
```

Next step

Now go to “Local host and repository files (iSeries, UNIX, and Windows)” on page 468 to continue the installation procedure.

Directory structure (Solaris)

The AMI tar file contains:

```
./amt/amtsdfts.tst : WebSphere MQ mqsc command file to create default WebSphere MQ
                     objects required by the AMI

./amt/amthost.xml : Sample AMI XML file used as the default host file

./amt/amt.dtd : AMI Document Type Definition file on which the AMI
                repository is based

./amt/ipla : The International Program License Agreement file
./amt/li : The License Information file

./amt/inc
   amtc.h : The C header file for the AMI
   amtcpp.hpp : The C++ header file for the AMI
   amtphlc.h: The C header file for the policy handler interface
   amtphmqc.h: The C header file for the policy handler interface (for WebSphere MQ)
   amtxc.h : The C header file for the extended AMI functions
   oamasami.h : The C header file for the OAMAS AMI subset

./amt/handlers
   libamtsphlr.so : C policy handler sample library

./amt/ldap/
   amtad.vbs : AMI Active Directory Visual Basic script file
   amtad.ldf : AMI Active Directory schema file
   amtsw.ldif : AMI SecureWay schema file for AIX, iSeries, HP-UX, Solaris,
                and Windows
   amtsw390.ldif : AMI SecureWay schema file for OS/390 TDBM
   amtsw390.at : AMI SecureWay schema file for OS/390 RDBM
   amtsw390.oc : AMI SecureWay schema file for OS/390 RDBM

./bin/amtldup : The AMI LDAP directory update program

./java/lib
   com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java

./lib
   libamt.so : The main AMI library
   libamtXML310.so : The AMI XML parsing library
   libamtCpp.so : The AMI C++ library
   libamtJava.so: The AMI JNI library
   libamtICUUC140.so : The AMI codepage translation library
   libamtICUDATA.so : The AMI codepage translation data library
   amtcmqm : Dynamic binding stub for WebSphere MQ Server library
   amtcmqic : Dynamic binding stub for WebSphere MQ Client library
   libamtldap_r.so : The AMI LDAP library

./amt/samp
   amtsamp.tst : WebSphere MQ mqsc command file to create WebSphere MQ objects
                 required by AMI samples
   amt.xml : Sample AMI XML repository for use with the AMI samples


./amt/samp/C
   amtsosnd.c : C source for object-level send and forget sample
   amtsorcv.c : C source for object-level receiver sample
   amtsoclt.c : C source for object-level client sample
   amtsosvr.c : C source for object-level server sample
   amtsopub.c : C source for object-level publisher sample
   amtsosub.c : C source for object-level subscriber sample
   amtsofsn.c : C source for object-level send file sample
   amtsofrc.c : C source for object-level receive file sample
   amtsosgs.c : C source for object-level send group sample
   amtsosgr.c : C source for object-level receive group sample
   amtshsnd.c : C source for high-level send and forget sample
   amtshrcv.c : C source for high-level receiver sample
   amtshclt.c : C source for high-level client sample
   amtshsvr.c : C source for high-level server sample
   amtshpub.c : C source for high-level publisher sample
   amtshsub.c : C source for high-level subscriber sample
   amtshfsn.c : C source for high-level send file sample
   amtshfrc.c : C source for high-level receive file sample

./amt/samp/C/bin
   amtsosnd : C object-level send and forget sample program
   amtsorcv : C object-level receiver sample program
   amtsoclt : C object-level client sample program
   amtsosvr : C object-level server sample program
   amtsopub : C object-level publisher sample program
   amtsosub : C object-level subscriber sample program
   amtsofsn : C object-level send file sample program
   amtsofrc : C object-level receive file sample program
   amtsosgs : C object-level send group sample program
   amtsosgr : C object-level receive group sample program
   amtshsnd : C high-level send and forget sample program
   amtshrcv : C high-level receiver sample program
   amtshclt : C high-level client sample program
   amtshsvr : C high-level server sample program
   amtshpub : C high-level publisher sample program
   amtshsub : C high-level subscriber sample program
   amtshfsn : C high-level send file sample program
   amtshfrc : C high-level receive file sample program

 ./amt/samp/C/handlers
   amtsphlh.h : C header file for policy handler sample
   amtsphlr.c : C source for policy handler sample
   amtsphlr.exp : C export file for policy handler sample

./amt/samp/Cpp
   SendForget.cpp : C++ source for send and forget sample
   Receiver.cpp : C++ source for receiver sample
   Client.cpp : C++ source for client sample
   Server.cpp : C++ source for server sample
   Publisher.cpp : C++ source for publisher sample
   Subscriber.cpp : C++ source for subscriber sample
   RcvFile.cpp : C++ source for receive file sample
   SendFile.cpp : C++ source for send file sample

./amt/samp/Cpp/bin
```

```
   SendForget : C++ send and forget sample program
   Receiver : C++ receiver sample program
   Client : C++ client sample program
   Server : C++ server sample program
   Publisher : C++ publisher sample program
   Subscriber : C++ subscriber sample program
   RcvFile : C++ source for receive file sample
   SendFile : C++ source for send file sample

./amt/samp/java
   SendForget.java : Java source for send and forget sample
   Receiver.java : Java source for receiver sample
   Client.java : Java source for client sample
   Server.java : Java source for server sample
   Publisher.java : Java source for publisher sample
   Subscriber.java : Java source for subscriber sample
   RcvFile.java : Java source for receive file sample
   SendFile.java : Java source for send file sample

./amt/samp/java/bin
   com.ibm.mq.amt.samples.jar : The jar file containing the AMI
     samples class files for Java
```

Installation on Windows

The AMI package for Windows 98, Windows NT, Windows Me, and Windows 2000 comes as a zip file, `ma0f_nt.zip`. Once unzipped it comprises:

readme	A file containing any product and information updates that have become available since this documentation was produced
setup	InstallShield installation program for WebSphere MQ AMI

In addition, it contains files used by the **setup** program.

Installation

1. Create an empty directory called `tmp` and make it current.
2. Store the `ma0f_nt.zip` file in this directory.
3. Uncompress it into `tmp` using Info-ZIP's UnZip program (or other unzip program).
4. Run **setup**.
5. Delete the `tmp` directory.

The files and directories created are listed in "Directory structure (Windows)" on page 462. The default location of these files depends on the version of the product you have installed. For a new installation of IBM WebSphere MQ for Windows Version 5.3, it is `C:\Program Files\IBM\WebSphere MQ`. If you have an earlier version, or an upgrade to V5.3 of an earlier version, it might be `C:\Program Files\MQSeries` or `C:\Program Files\IBM\MQSeries`. In either case, a different directory might have been specified when the product was installed.

Removing the AMI

To uninstall the Application Messaging Interface, use the Add/Remove Programs control panel.

Note: If you specified a directory other than the default during installation, you must remove this directory from the PATH environment variable. In addition, if you used Java, you must remove the AMI entries from the CLASSPATH. For example:

```
C:\Program Files\IBM\WebSphere MQ\java\lib\com.ibm.mq.amt.jar;
C:\Program Files\IBM\WebSphere MQ\amt\samples\java\bin\
  com.ibm.mq.amt.samples.jar;
```

These will not be removed by Add/Remove Programs. (See "Setting the runtime environment".)

Setting the runtime environment

By default, the location of the AMI runtime binary files matches that of WebSphere MQ (for example `C:\Program Files\IBM\WebSphere MQ\bin`). If you specified a different directory for the runtime files, you **must** add it to the PATH environment variable.

To use the samples, add the sample C and C++ binary directories to your PATH environment variable. For example (assuming that the root directory for WebSphere MQ is `C:\Program Files\IBM\WebSphere MQ`):

```
set PATH=%PATH%;C:\Program Files\IBM\WebSphere MQ\amt\samples\C\bin;
            C:\Program Files\IBM\WebSphere MQ\amt\samples\Cpp\bin;
```

Installation on Windows

When running Java, the AMI classes and samples must be contained in the CLASSPATH environment variable. For example:

```
C:\Program Files\IBM\WebSphere MQ\java\lib\com.ibm.mq.amt.jar;
C:\Program Files\IBM\WebSphere MQ\amt\samples\java\bin\
  com.ibm.mq.amt.samples.jar;
```

This is done by the **setup** program.

> **Next step**
>
> Now go to “Local host and repository files (iSeries, UNIX, and Windows)” on page 468 to continue the installation procedure.

Directory structure (Windows)

On Windows platforms, the directory structure contains:

```
.\amt\amtsdfts.tst : WebSphere MQ mqsc command file to create default WebSphere MQ
                     objects required by the AMI

.\amt\amthost.xml : Sample AMI XML file used as the default host file

.\amt\amt.dtd : AMI Document Type Definition file on which the AMI
                repository is based

.\amt\ipla : The International Program License Agreement file
.\amt\li : The License Information file

.\amt\include
   amtc.h : The C header file for the AMI
   amtcpp.hpp : The C++ header file for the AMI
   amtphlc.h: The C header file for the policy handler interface
   amtphmqc.h: The C header file for the policy handler interface (for WebSphere MQ)
   amtxc.h : The C header file for the extended AMI functions
   oamasami.h: The C header file for the OAMAS AMI subset

.\amt\handlers
   amtsphlr.dll : C policy handler sample library

.\amt\ldap\
   amtad.vbs : AMI Active Directory Visual Basic script file
   amtad.ldf : AMI Active Directory schema file
   amtsw.ldif : AMI SecureWay schema file for AIX, iSeries, HP-UX, Solaris,
                and Windows
   amtsw390.ldif : AMI SecureWay schema file for OS/390 TDBM
   amtsw390.at : AMI SecureWay schema file for OS/390 RDBM
   amtsw390.oc : AMI SecureWay schema file for OS/390 RDBM

.\java\lib
   com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java

.\bin
   amt.dll : The main AMI library
   amtXML310.dll : The AMI XML parsing library
   amtCpp.dll : The AMI C++ library
   amtJava.dll : The AMI JNI library
   amtICUUC140.dll : The AMI codepage translation library
   amtICUDATA.dll : The AMI codepage translation data library
   amtldap.dll : The AMI LDAP library
   amtldup.exe : The AMI LDAP directory update program
```

```
.tools\lib
   amt.lib : The AMI LIB file used for building C programs
   amtCpp.lib : The AMI LIB file used for building C++ programs

.\amt\samples
   amtsamp.tst : WebSphere MQ mqsc command file to create WebSphere MQ objects
                 required by AMI samples
   amt.xml : Sample AMI XML repository for use with the AMI samples

.\amt\samples\C
   amtsosnd.c : C source for object-level send and forget sample
   amtsorcv.c : C source for object-level receiver sample
   amtsoclt.c : C source for object-level client sample
   amtsosvr.c : C source for object-level server sample
   amtsopub.c : C source for object-level publisher sample
   amtsosub.c : C source for object-level subscriber sample
   amtsofsn.c : C source for object-level send file sample
   amtsofrc.c : C source for object-level receive file sample
   amtsosgs.c : C source for object-level send group sample
   amtsosgr.c : C source for object-level receive group sample
   amtshsnd.c : C source for high-level send and forget sample
   amtshrcv.c : C source for high-level receiver sample
   amtshclt.c : C source for high-level client sample
   amtshsvr.c : C source for high-level server sample
   amtshpub.c : C source for high-level publisher sample
   amtshsub.c : C source for high-level subscriber sample
   amtshfsn.c : C source for high-level send file sample
   amtshfrc.c : C source for high-level receive file sample

.\amt\samples\C\bin
   amtsosnd.exe : C object-level send and forget sample program
   amtsorcv.exe : C object-level receiver sample program
   amtsoclt.exe : C object-level client sample program
   amtsosvr.exe : C object-level server sample program
   amtsopub.exe : C object-level publisher sample program
   amtsosub.exe : C object-level subscriber sample program
   amtsofsn.exe : C object-level send file sample program
   amtsofrc.exe : C object-level receive file sample program
   amtsosgs.exe : C object-level send group sample program
   amtsosgr.exe : C object-level receive group sample program
   amtshsnd.exe : C high-level send and forget sample program
   amtshrcv.exe : C high-level receiver sample program
   amtshclt.exe : C high-level client sample program
   amtshsvr.exe : C high-level server sample program
   amtshpub.exe : C high-level publisher sample program
   amtshsub.exe : C high-level subscriber sample program
   amtshfsn.exe : C high-level send file sample program
   amtshfrc.exe : C high-level receive file sample program

.\amt\samples\Cpp
   SendForget.cpp : C++ source for send and forget sample
   Receiver.cpp : C++ source for receiver sample
   Client.cpp : C++ source for client sample
   Server.cpp : C++ source for server sample
   Publisher.cpp : C++ source for publisher sample
   Subscriber.cpp : C++ source for subscriber sample
   RcvFile.cpp : C++ source for receive file sample
   SendFile.cpp : C++ source for send file sample

.\amt\samples\Cpp\bin
   SendForget.exe : C++ send and forget sample program
   Receiver.exe : C++ receiver sample program
   Client.exe : C++ client sample program
   Server.exe : C++ server sample program
   Publisher.exe : C++ publisher sample program
   Subscriber.exe : C++ subscriber sample program
   RcvFile.exe : C++ receive file sample program
   SendFile.exe : C++ send file sample program

.\amt\samples\handlers
   amtsphlh.h : C header file for policy handler sample
   amtsphlr.c : C source for policy handler sample
   amtsphlr.def : C definition file for policy handler sample

.\amt\samples\java
   SendForget.java : Java source for send and forget sample
   Receiver.java : Java source for receiver sample
   Client.java : Java source for client sample
   Server.java : Java source for server sample
   Publisher.java : Java source for publisher sample
   Subscriber.java : Java source for subscriber sample
   RcvFile.java : Java source for receive file sample
   SendFile.java : Java source for send file sample

.\amt\samples\java\bin
   com.ibm.mq.amt.samples.jar : The jar file containing the AMI
     samples class files for Java
```

Installation on z/OS

The AMI is installed automatically with WebSphere MQ for z/OS.

Installation

The files and directories created are listed in "Directory structure (z/OS)".

Setting the runtime environment

Batch and RRS-batch

Make sure that the location of the AMI runtime library is added to your JCL STEPLIB concatenation.

IMS

Make sure that the location of the AMI runtime library is added to your IMS message processing region JCL STEPLIB concatenation.

CICS

Make sure that the location of the AMI runtime library is added to your region's DFHRPL concatenation, and the AMI library is defined in your CICS CSD. A sample CSD script, inhlq.SCSQPROC(AMTCSD10), is supplied to help define the AMI library to CICS.

Next step

Now go to "Local host and repository files (z/OS)" on page 470 to continue the installation procedure.

Directory structure (z/OS)

On z/OS platforms, the directory structure contains the following (where 'hlq' is the high-level qualifier of the AMI installation):

```
hlq.SCSQLOAD
  AMTBL10  : The main AMI library (batch)
  AMTCL10  : The main AMI library (CICS)
  AMTIL10  : The main AMI library (IMS)
  AMTRL10  : The main AMI library (RRS-batch)
  AMTBS10  : Stub to build COBOL applications (batch)
  AMTCS10  : Stub to build COBOL applications (CICS)
  AMTIS10  : Stub to build COBOL applications (IMS)
  AMTRS10  : Stub to build COBOL applications (RRS-batch)
  AMTASM10 : Repository cache generator
  AMTBURV  : Print the repository
  AMTLDAPL : LDAP parser DLL


hlq.SCSQANLE
  AMTMSE10 : US English messages
  AMTMSG10 : US English messages

hlq.SCSQANLU
  AMTMSG10 : Uppercase US English messages
  AMTMSU10 : Uppercase US English messages

hlq.SCSQANLK
  AMTMSG10 : Kanji messages
  AMTMSK10 : Kanji messages

hlq.SCSQANLC
```

```
  AMTMSG10 : Chinese messages
  AMTMSC10 : Chinese messages

hlq.SCSQC370
  AMTC : The C header file for the AMI
  AMTSPHLH : The C header file for policy handler sample
  AMTPHLC  : Policy handler interface definition
  AMTPHMQC : Transport-specific policy handler definitions for WebSphere MQ
  AMTXC    : AMI extensions for policy handler callback functions

hlq.SCSQCOBC
  AMTELEML : COBOL copybook for the AMELEM structure
  AMTELEMV : COBOL copybook for the AMELEM structure, with default values
  AMTV     : The main COBOL copybook for the AMI

hlq.SCSQPROC
  AMT      : Sample AMI XML repository for use with the AMI samples.
  AMTCSD10 : CICS definitions for the AMI library.
  AMTHOST  : Sample AMI XML file for use as the default host file (UTF-8).
  AMTHOST2 : Sample AMI XML file for use as the default host file
             (EBCDIC 1047).
  AMTSDFTS : WebSphere MQ mqsc command file to create default WebSphere MQ objects
             required by the AMI.
  AMTSAMP  : WebSphere MQ mqsc command file to create WebSphere MQ objects required
             by AMI samples.

hlq.SCSQDEFS
  AMTBD10  : DLL side-deck to build C applications (batch)
  AMTCD10  : DLL side-deck to build C applications (CICS)
  AMTRD10  : DLL side-deck to build C applications (RRS-batch)
  AMTID10  : DLL side-deck to build C applications (IMS)


hlq.SCSQCOBS (COBOL samples for Batch, RRS, CICS, and IMS)
  AMTVHSND : COBOL source for high-level send and forget sample
  AMTVHRCV : COBOL source for high-level receiver sample
  AMTVHCLT : COBOL source for high-level client sample
  AMTVHSVR : COBOL source for high-level server sample
  AMTVHPUB : COBOL source for high-level publisher sample
  AMTVHSUB : COBOL source for high-level subscriber sample
  AMTVHFSN : COBOL source for high-level group send file transfer sample
  AMTVHFRC : COBOL source for high-level group receive file transfer sample
  AMTVOSND : COBOL source for object-level send and forget sample
  AMTVORCV : COBOL source for object-level receiver sample
  AMTVOCLT : COBOL source for object-level client sample
  AMTVOSVR : COBOL source for object-level server sample
  AMTVOPUB : COBOL source for object-level publisher sample
  AMTVOSUB : COBOL source for object-level subscriber sample
  AMTVOSGS : COBOL source for object-level group send sample
  AMTVOSGR : COBOL source for object-level group receive sample
  AMTVOFSN : COBOL source for object-level send file transfer sample
  AMTVOFRC : COBOL source for object-level receive file transfer sample


hlq.SCSQC37S (C samples for Batch, RRS, CICS, and IMS)
  AMTSHSND : C source for high-level send and forget sample
  AMTSHRCV : C source for high-level receiver sample
  AMTSHCLT : C source for high-level client sample
  AMTSHSVR : C source for high-level server sample
  AMTSHPUB : C source for high-level publisher sample
  AMTSHSUB : C source for high-level subscriber sample
  AMTSHFSN : C source for high-level group send file transfer sample
  AMTSHFRC : C source for high-level group receive file transfer sample
  AMTSOSND : C source for object-level send and forget sample
  AMTSORCV : C source for object-level receiver sample
  AMTSOCLT : C source for object-level client sample
  AMTSOSVR : C source for object-level server sample
  AMTSOPUB : C source for object-level publisher sample
```

```
AMTSOSUB : C source for object-level subscriber sample
AMTSOSGS : C source for object-level group send sample
AMTSOSGR : C source for object-level group receive sample
AMTSOFSN : C source for object-level send file transfer sample
AMTSOFRC : C source for object-level receive file transfer sample
AMTSPHLR : C source for policy handler sample
```

Local host and repository files (iSeries, UNIX, and Windows)

The AMI uses a *repository file* and a *local host file*. Their location and names must be specified to the AMI.

Default location

On iSeries, the default directory for the files is:

```
/QIBM/UserData/mqm/amt
```

On UNIX, the default directory for the files is:

```
/usr/mqm/amt              (AIX)

/opt/mqm/amt              (HP-UX, Solaris)
```

On Windows, the default location is a directory called `\amt` under the directory in which WebSphere MQ is installed (see “Installation on Windows” on page 461). For example, if WebSphere MQ is installed in the `C:\Program Files\IBM\WebSphere MQ` directory, the default directory for the AMI data files on Windows is:

```
C:\Program Files\IBM\WebSphere MQ\amt
```

Default names

The default name for the repository file is `amt.xml`, and the default name for the host file is `amthost.xml`.

A sample host file (which can be used as a default) is provided in the correct location.

A sample repository file is located in the following directory:

```
/QIBM/ProdData/mqm/amt/samp (iSeries)

/amt/samp                   (UNIX)

\amt\samples                (Windows)
```

Overriding the default location and names

You can override where the AMI looks for the repository and local host files by using an environment variable:

```
ADDENVVAR ENVVAR(AMT_DATA_PATH) VALUE('/directory')  (iSeries)

export AMT_DATA_PATH = /directory                    (UNIX)

set AMT_DATA_PATH = X:\directory                     (Windows)
```

You can override the default names of the repository and local host files by using environment variables:

```
ADDENVVAR ENVVAR(AMT_REPOSITORY) VALUE('myData.xml') (iSeries)
ADDENVVAR ENVVAR(AMT_HOST) VALUE('myHostFile.xml')

export AMT_REPOSITORY = myData.xml                   (UNIX)
export AMT_HOST = myHostFile.xml

set AMT_REPOSITORY = myData.xml                      (Windows)
set AMT_HOST = myHostFile.xml
```

The directories `intlFiles` and `locales`, and the `.txt` and `.cnv` files in the `locales` directory, must be located relative to the directory containing the local host file. This applies whether you are using the default directory or have overridden it as described previously.

In C++ and Java, there is an extra level of flexibility in setting the location and names of the repository and local host files. You can specify the directory in which they are located by means of a name in the constructor of the AmSessionFactory class:

```
AmSessionFactory(name);
```

This name is equivalent to the AMT_DATA_PATH environment variable. If set, the name of the AmSessionFactory takes precedence over the AMT_DATA_PATH environment variable.

The repository and local host file names can be set using methods of the AmSessionFactory class:

```
setRepository(name);
setLocalHost(name);
```

These AmSessionFactory methods take precedence over the AMT_REPOSITORY and AMT_HOST environment variables.

Once an AmSession has been created using an AmSessionFactory, the repository and local host file names and location are set for the complete life of that AmSession.

Local host file

An AMI installation must have a local host file. It defines the mapping from a connection name (default or repository defined) to the name of the WebSphere MQ queue manager that you want to connect to on your local machine.

If you are not using a repository, or are opening (or initializing) a session using a policy that does not define a connection, the connection name is assumed to be `defaultConnection`. Using the sample `amthost.xml` file, as shown below, this maps to an empty string that defines a connection with the default queue manager.

```
<?xml version="1.0" encoding="UTF-8"?>
<queueManagerNames
        defaultConnection = ""
        connectionName1   = "queueManagerName1"
        connectionName2   = "queueManagerName2"
/>
```

To change the default connection to a named queue manager of your choice, such as ‘QMNAME’, edit the local host file to contain the following string:

```
defaultConnection = "QMNAME"
```

If you want a repository defined connection name, such as `connectionName1`, to provide a connection to queue manager ‘QMNAME1’, edit the local host file to contain the following string:

```
connectionName1   = "QMNAME1"
```

The repository connection names are not limited to the values shown (`connectionName1` and `connectionName2`). Any name can be used provided it is unique in both the repository and local host files, and consistent between the two.

Repository file

You can operate an AMI installation with or without a repository file. If you are using a repository file, such as the sample `amt.xml` file, you must have a corresponding `amt.dtd` file in the same directory (the local host file must be in this directory as well).

The repository file (or an LDAP directory service) provides definitions for policies and services. If you do not use a repository file (or an LDAP directory service), AMI uses its built-in definitions. For more information, see Chapter 18, “Defining services, policies, and policy handlers” on page 491.

For information about LDAP support, see Chapter 19, “Lightweight Directory Access Protocol support” on page 505.

Local host and repository files (z/OS)

The AMI uses a *repository file* and a *local host file*. Their location and names must be specified to the AMI.

Batch, RRS-batch, IMS

The repository file is optional, and the host file is mandatory. Sample repository and host files are installed to `hlq.SCSQPROC`.

By default, the AMI uses the DD name `AMT` (within your job or IMS message processing region JCL) to locate the repository file, and the DD name `AMTHOST` to locate the host file.

Because the repository and host files are located using DD statements in your job or IMS message processing region JCL, you can choose which files to use without using environment variables. If you do want to use environment variables, you can override the locations of these files using the Language Environment ENVAR Run-Time Option.

An example PARM statement for a C application, which changes the DD names used for the repository and local host files, is:

```
PARM=('ENVAR(AMT_REPOSITORY=DD:MYREPOS,AMT_HOST=DD:MYHOST) / ARGS')
```

An example PARM statement for a COBOL application, which changes the DD name used for the repository and local host files, is:

```
PARM=('ARGS / ENVAR(AMT_REPOSITORY=DD:MYREPOS,AMT_HOST=DD:MYHOST)')
```

In both these examples, `ARGS` indicates the program’s arguments. See the *z/OS Language Environment Programming Guide* for more information about Language Environment Run-Time Options.

CICS

Under CICS, the AMI does not need a local host file, and the repository file is optional. To use the sample repository file under CICS, copy the repository into a VSAM entry-sequenced dataset using the IDCAMS utilities.

By default, the AMI uses a CICS FILE definition called `AMT` to locate the repository file.

As the repository is located using a CICS FILE definition, you can change which file to use by changing that definition. You can also change the CICS file name using environment variables and the z/OS C/C++ function `setenv()`:

```
setenv( "AMT_REPOSITORY", "NAME", 1 );
```

Local host file

An AMI installation using z/OS batch, IMS, or RRS-batch must have a local host file. It defines the mapping from a connection name (default or repository defined) to the name of the WebSphere MQ queue manager that you want to connect to on your z/OS installation. (The local host file is not needed for CICS, because there is only one WebSphere MQ queue manager that a given CICS region can connect to).

If you are not using a repository, or are opening (or initializing) a session using a policy that does not define a connection, the connection name is assumed to be `defaultConnection`. Using the sample `AMTHOST` file, as shown below, this maps to an empty string that defines a connection with the default queue manager.

Note: The AMTHOST file shown below is an UTF-8 text file best suited to editing on a workstation. If you prefer to maintain your host file on the host, you should use the AMTHOST2 sample, which is in an EBCDIC codepage.

```
<?xml version="1.0" encoding="UTF-8"?>
<queueManagerNames
        defaultConnection = ""
        connectionName1   = "queueManagerName1"
        connectionName2   = "queueManagerName2"
/>
```

To change the default connection to a named queue manager of your choice, such as ‘QMNAME’, edit the local host file to contain the following string:

```
defaultConnection = "QMNAME"
```

If you want a repository defined connection name, such as `connectionName1`, to provide a connection to queue manager ‘QMNAME1’, edit the local host file to contain the following string:

```
connectionName1   = "QMNAME1"
```

The repository connection names are not limited to the values shown (`connectionName1` and `connectionName2`). Any name can be used provided it is unique in both the repository and local host files, and consistent between the two.

“Repository and local host caches” on page 472 explains how to use a local host cache instead of a local host file.

Repository file

You can operate an AMI installation with or without a repository file. The repository file provides definitions for policies and services. If you do not use a
| repository file, AMI uses its built-in definitions. Use the AMI Administration Tool
| to create your own repository file and then use FTP to transfer the file in binary
| mode to z/OS. The format of the repository file on z/OS is not crucial. For
| example, you can use Record Format FB, Record Length 80, as a member of a PDS.
For more information, see Chapter 18, “Defining services, policies, and policy handlers” on page 491.

“Repository and local host caches” on page 472 explains how to use a repository cache instead of a repository file.

Repository and local host caches

On z/OS, you can generate caches for use instead of repository and local host files. This gives a higher performance alternative to the files, but requires some additional configuration.

Generating caches

The AMI on z/OS includes a program (AMTASM10) that generates assembler source code defining repository and local host caches. This program runs in a similar manner to any AMI batch program, and outputs a repository cache definition to the DD name ASMREPOS, and a local host cache to the DD name ASMHOST. The cache generator issues messages to the SYSPRINT data set, and returns zero if it is successful.

The cache can be generated from the source XML file or from definitions that are stored in an LDAP directory.

Building a cache from source xml files: Specify the input xml repository data file with a DD statement for the file AMT. Specify the input host data file with a DD statement for the file AMTHOST.

Here is a sample JCL fragment to run the cache generator (with US English messages):

```
//GO EXEC PGM=AMTASM10
//STEPLIB  DD DSN=hlq.SCSQLOAD,DISP=SHR
//         DD DSN=hlq.SCSQANLE,DISP=SHR
//AMTHOST  DD DSN=hlq.SCSQPROC(AMTHOST),DISP=SHR
//AMT      DD DSN=hlq.SCSQPROC(AMT),DISP=SHR
//SYSPRINT DD SYSOUT=*
//ASMHOST  DD DSN=target(AMTHOST),DISP=SHR
//ASMREPOS DD DSN=target(AMT),DISP=SHR
```

When you have generated assembler source code successfully for your repository and host file cache, you must assemble and link edit them. Messages that the cache generator returns are described in “Cache generator messages” on page 473.

Building a cache from the LDAP directory: To build a cache from the LDAP directory, several environment variables must be set before the AMTASM10 cache build program is invoked.

AMT_LDAP_USE must be set to YES. This makes the AMTASM10 cache build program use the LDAP directory when it builds the cache. Other required environment variables are listed in Chapter 19, “Lightweight Directory Access Protocol support” on page 505.

Here is a sample JCL fragment to build a repository cache from an LDAP directory:

```

//GEN     EXEC PGM=AMTASM10,
// PARM='POSIX(ON),ENVAR(_CEE_ENVFILE=DD:MYVARS,)/'
//STEPLIB   DD DSNAME=hlq.SCSQLOAD,DISP=SHR
//          DD DSNAME=hlq.SCSQANLE,DISP=SHR
//          DD DSNAME=hlqforLE.SCEERUN,DISP=SHR
//          DD DISP=SHR,DSN=your.system.ldap.SGLDLNK
//AMTHOST   DD DSN=&HOSTIN,DISP=SHR
//ASMHOST   DD DSN=target(AMTHOST)
//ASMREPOS  DD DSN=target(AMT)
//MYVARS      DD *
AMT_LDAP_USE=YES
```

```
AMT_REPOSITORY=ldap://your.host:your_port_number/Base Dn
AMT_LDAP_BINDDN="bind_distinguished_name"
AMT_TRACE_LDAP=YES
AMT_TRACE_LEVEL=9
```

The environment variables AMT_LDAP_USE and AMT_REPOSITORY are mandatory, but AMT_LDAP_BINDDN, AMT_TRACE_LDAP, and AMT_TRACE_LEVEL are optional. For information about these environment varables, and other AMT_LDAP_* environment variables required, refer to Chapter 19, “Lightweight Directory Access Protocol support” on page 505.

Cache generator messages

The following messages are issued by the cache generator. Terms like “%li” will be printed as decimal numbers; they hold the AMI completion and reason codes.

```
"AMT0001W AMI MESSAGE MODULE NOT FOUND"

/**********************************************************************/
/* Explanation:                                                       */
/*  The AMI failed to load its message module.                        */
/* User Response:                                                     */
/*  Batch, IMS: Ensure that one of the language-specific datasets is  */
/*              in your STEPLIB concatenation.                        */
/*  CICS:       Ensure that one of the language-specific datasets is  */
/*              in your DFHRPL concatenation, and the message module  */
/*              AMTMSG10 is defined to CICS.                          */
/**********************************************************************/
```

```
"AMT0002W AMI failure, AMCC=%li, AMRC=%li"

/*********************************************************************/
/* Explanation:                                                      */
/*   An AMI operation failed.                                        */
/* User Response:                                                    */
/*   See the WebSphere MQ Application Messaging Interface Manual     */
/*   for an explanation of CompCode, AMCC, and Reason, AMRC.         */
/*********************************************************************/
```

```
"AMT0003I AMI repository cache warning, AMCC=%li, AMRC=%li"

/*********************************************************************/
/* Explanation:                                                      */
/*   An AMI operation generated a warning.                           */
/* User Response:                                                    */
/*   See the WebSphere MQ Application Messaging Interface Manual     */
/*   for an explanation of CompCode, AMCC, and Reason, AMRC.         */
/*********************************************************************/
```

```
"AMT0004I AMI repository cache created from %s"

/*********************************************************************/
/* Explanation:                                                      */
/*   A repository cache was successfully created from either         */
/*    the default settings, an XML file, LDAP or the existing        */
/*    cache is being used.                                           */
/*    %s will be printed as either DEFAULT, XML FILE, LDAP           */
/*    or EXISTING CACHE                                              */
/* User Response:                                                    */
/*   None.                                                           */
/*********************************************************************/
```

Local host and repository files (z/OS)

```
"AMT0005I AMI host file cache created"

/*******************************************************************/
/* Explanation:                                                    */
/*   A host file cache was successfully created.                   */
/* User Response:                                                  */
/*   None.                                                         */
/*******************************************************************/
```

```
"AMT0006I AMI Repository View Report complete"

/*******************************************************************/
/* Explanation:                                                    */
/*   The repository view Report was successfully created.          */
/* User Response:                                                  */
/*   None.                                                         */
/*******************************************************************/
```

```
"AMT0007W AMI New Cache not generated"

/*******************************************************************/
/* Explanation:                                                    */
/*   A new cache was not generated. The existing cache             */
/*   is still available for use.                                   */
/* User Response:                                                  */
/*   None.                                                         */
/*******************************************************************/
```

Using a cache

When your application creates an AMI session, the AMI first tries to load caches, before it tries to open files. The module that the AMI loads has the same name as the corresponding filename, that is `AMT` for the repository file and `AMTHOST` for the local host file. You can modify the name that will be loaded using environment variables as discussed in "Batch, RRS-batch, IMS" on page 470 and "CICS" on page 470.

Batch, RRS-batch, and IMS applications must include the dataset that contains your cache in the JCL STEPLIB. There is no need to use DD AMT or DD AMTHOST statements to locate the cached files.

CICS applications must add the dataset that contains the cache to the region DFHRPL, and define the cache to CICS using the CICS supplied CEDA transaction. There is no need to define the AMT file to CICS. The AMTHOST file is not applicable for CICS.

Viewing the repository

On z/OS, you can use the Repository View utility, AMTBURV, to view the contents of the repository. The utility displays either the contents of the repository cache or the contents of the repository file. If the library holding the cache is concatenated in the steplib, its contents are displayed, otherwise the contents of the repository file (DD statement AMT) are displayed.

Here is a sample JCL fragment to run the repository view utility:

```
//GEN     EXEC PGM=AMTBURV,
//STEPLIB   DD DSNAME=hlq.SCSQLOAD,DISP=SHR
//          DD DSNAME=hlq.SCSQANLE,DISP=SHR
//          DD DSNAME=hlqforLE.SCEERUN,DISP=SHR
//          DD DSNAME=your.generated.cache,DISP=SHR
//AMTHOST   DD DSN=your.amthost.xml.file
//AMT       DD DSN=your.amt.xml.file
//REPORT    DD DSN=report.output.file
//SYSPRINT  DD  SYSOUT=*
```

AMTBURV requires data definition (DD) statements with the following names:

AMTHOST	This statement names the local host file. It is required only if the generated cache is not used.
AMT	This statement names the repository file. It is required only if the generated cache is not used
REPORT	This statement names the dataset for output. It is a variable length record with a logical record length (LRECL) of 1028 and a blocksize(BLKSIZE) of 6144. The report output can also be written to sysout.

A sample fragment from the Repository View utility is as follows:

```
***************SERVICE OBJECTS***************
Element Name is ServicePoint
Attribute Name is xmi.id, Attribute Value is 0406000a-0900-0000-0000-09040a0d000
Attribute Name is xmi.label, Attribute Value is AMT.SAMPLE.SUBSCRIBER
Element Name is ServicePoint_queueName
Attribute Name is xmi.label, Attribute Value is queueName
Attribute Name is value, Attribute Value is SYSTEM.BROKER.CONTROL.QUEUE
Element Name is ServicePoint_queueMgrName
Attribute Name is xmi.label, Attribute Value is queueMgrName
Attribute Name is value, Attribute Value is
```

The AMI Administration Tool

The AMI Administration Tool is for use on Windows NT and Windows 2000 only.

Installation

The AMI Administration Tool is packaged with the AMI in `ma0f_nt.zip` and optionally installed with the AMI using the setup InstallShield program (see “Installation on Windows” on page 461). It is installed in sub-directory `amt\AMITool`.

To start the AMI Administration Tool, select **IBM WebSphere MQ AMI → IBM WebSphere MQ AMI Administration Tool** using the **Start Programs menu**, or double-click on the file `\amt\AMITool\amitool.bat`.

To verify that the tool has been installed correctly, click on **Open** in the **File** menu, navigate to the `\amt\samples` directory, and open the sample repository file `amt.xml`. You should see a number of services, policies, and policy handlers in the navigation pane on the left. Select one of them by clicking on it, and you should see its attributes displayed in the pane on the right.

Operation

The AMI Administration Tool enables you to create definitions for:

Service points	used to create sender or receiver services
Distribution lists	must include at least one sender service
Publishers	must include a sender service as the broker service
Subscribers	must include sender and receiver services as the broker and receiver services
Policies	contain sets of attributes: initialization, general, send, receive, publish, subscribe, handler
Policy handlers	used to create policy handler library definitions

The default attributes provided by the tool are as specified in “Service definitions” on page 494, “Policy definitions” on page 497, and “Policy handler definitions” on page 504.

When you have entered the definitions you require, select **Save** in the **File** menu to save them as an XML-format repository file. It is recommended that you define all your services, policies, and policy handlers in the same repository file.

The repository file must be copied to a location where it can be accessed by the AMI (see “Local host and repository files (iSeries, UNIX, and Windows)” on page 468). If the Application Messaging Interface is on the same system as the tool, the repository file can be copied to the AMI directory. Otherwise, the repository file must be transferred to that system using a method such as file sharing or FTP.

Notes:

1. To open an existing repository file (including the `amt.xml` file provided in the samples directory), the repository file and the `amt.dtd` file must both be in the same directory.
2. If you are using the AMI Administration Tool to create an XML repository file for an earlier version of the AMI (for example, using the AMI 1.2 Administration Tool to create an XML file for use with AMI 1.1), then you must ensure that you include the newer `amt.dtd` file with the new XML file in the same directory, replacing the old `amt.dtd` file as necessary.

For further information, refer to the AMI Administration Tool online help, or Chapter 18, “Defining services, policies, and policy handlers” on page 491 and “Updating LDAP from a repository” on page 509.

Connecting to WebSphere MQ

You can connect to WebSphere MQ, the transport layer, using a WebSphere MQ server or a WebSphere MQ client. Using the default policy, the AMI automatically detects whether it should connect directly or as a client. If you have an installation that has both a WebSphere MQ client and a WebSphere MQ queue manager, and you want the AMI to use the client for its connection, you must specify the Connection Type as Client in the policy initialization attributes (see “Policy definitions” on page 497).

Using MQSeries Integrator Version 1

If you are using the AMI with MQSeries Integrator Version 1, the Service Type for the sender service point must be defined in the repository as ‘MQSeries Integrator V1’ (see “Service definitions” on page 494). This causes an MQRFH header containing application group and message type name/value elements to be added to a message when it is sent.

The Application Group definition is included in the policy send attributes (see “Policy definitions” on page 497). The message type is defined as the message format value set in the message object (using **amMsgSetFormat**, for example). If this is set to AMFMT_NONE, the message type is defined as the Default Format for the sender service point (a maximum of eight characters in WebSphere MQ). If you wish to specify the message type directly, you must do this explicitly using the **amMsgAddElement** function in C, or the equivalent **addElement** method in C++ and Java. This allows you to add a message type that differs from the message format, and is more than eight characters long.

Using MQSeries Publish/Subscribe

You can use the publish/subscribe functions of the AMI with MQSeries Publish/Subscribe (see the *MQSeries Publish/Subscribe User's Guide*). The Service Type for the sender and receiver service points used by the publisher and subscriber must be defined in the repository as MQRFH (see "Service definitions" on page 494). This causes an MQRFH header containing publish/subscribe name/value elements to be added to a message when it is sent.

Using WebSphere MQ Integrator Version 2

You can use your existing AMI repository file, MQSeries Publish/Subscribe applications, and MQSeries Integrator Version 1 (MQSI V1) applications unchanged with WebSphere MQ Integrator Version 2 (WMQI V2).

Alternatively, if you are writing a new application or wish to exploit some of the additional function provided by WMQI V2, you should specify 'WebSphere MQ Integrator V2' or 'RF Header V2' for the Service Type of 'Service Points' in your repository file. This is accomplished using the AMI Administration Tool.

The AMI makes it easy for applications to send messages to and receive messages from WMQI V2 and to exploit its publish and subscribe functions.

Applications send messages to WMQI V2 using the standard AMI send verbs. If the service point is defined as a Service Type of 'WebSphere MQ Integrator V2', the AMI automatically builds an MQRFH2 header at the beginning of the message and adds the default MCD parameters from the Service point definition if they have been defined. An application can therefore be unaware that it is communicating with WMQI V2. Applications requiring more control can explicitly add the MCD information using the **amMsgAddElement** C, **AMSADEL** COBOL, or **AmMessage::addElement** C++ and Java calls. The default MCD values are ignored if the application has added the elements to the message explicitly. The MQRFH2 and MCD fields are described in the *WebSphere MQ Integrator Version 2 Programming Guide*.

Publish/subscribe applications use the standard publish, subscribe and unsubscribe calls. However, subscribing applications can exploit content-based publish/subscribe by passing a filter on subscribe and unsubscribe calls. The syntax of the filter string is described in the *WebSphere MQ Integrator Version 2 Programming Guide*.

If you specify the Service Type as 'RF Header V2', the AMI selects and uses the Publish and Subscribe policy options applicable to WMQI V2 when sending publish, subscribe, and unsubscribe requests to the broker. Default MCD field values are ignored and not included in the message.

If you specify the Service Type as 'WebSphere MQ Integrator V2', the AMI selects and uses the Publish and Subscribe policy options that are applicable to WMQI V2 when sending publish, subscribe and unsubscribe requests. In addition, the AMI inserts each of the following values into any message being sent using this service point where a non-blank default value has been specified for the item concerned (in the Service Point Default MCD value) and the item has not been explicitly added by the application:

```
message service domain (Default MCD Domain)
message set (Default MCD Set)
message type (Default MCD Type)
message format (Default MCD Format)
```

Connecting to WebSphere MQ

If you wish to perform content-based publish/subscribe operations using WMQI V2, one or more filters must be specified and added to the messages used with subscribe requests. A filter can be added to a subscribe (and unsubscribe) message by specifying the filter as a parameter with the high-level subscribe (and unsubscribe) functions in C and COBOL or by using add filter calls before calling subscribe (or unsubscribe).

Note that in addition to add filter, there are delete filter, get filter and get filter count functions available for filter manipulation.

When a broker response message is received for a Publish or Subscribe request, an AMMSGTNE get named element call specifying the name as AMPS_COMP_CODE will always return a value corresponding to one of the following constants:

- AMPS_CC_OK
- AMPS_CC_WARNING
- AMPS_CC_ERROR

The value is returned whether the response originated from MQSeries Publish/Subscribe or WebSphere MQ Integrator Version 2. This allows the broker to recognize the broker being used. The AMI performs the required mapping of WebSphere MQ Integrator Version 2 response values as necessary.

Migrating to WebSphere MQ Integrator V2 from V1 and MQSeries Publish/Subscribe

WebSphere MQ Integrator V2 supports applications written to use MQSeries Integrator V1 and MQSeries Publish/Subscribe. Existing AMI applications and the Service Type in the repository Service Point definitions do not therefore need to be changed.

Applications that want to exploit new functions in WebSphere MQ Integrator V2 should have their Service Point definitions changed to a Service Type of ‘WebSphere MQ Integrator V2’ and, if necessary, use the new AMI calls and parameters.

Existing publish/subscribe applications that have used the element calls to explicitly add name value pairs to the MQRFH can continue to use the same names for the elements when migrating to WebSphere MQ Integrator V2.

Creating default WebSphere MQ objects

The Application Messaging Interface makes use of default WebSphere MQ objects, which must be created before using the AMI. To do this, you run the MQSC script `amtsdfts.tst` (you might want to edit this file first, to suit the requirements of your installation).

For iSeries, start the local queue manager by typing the following on the command line, where `QMName` is the name of your WebSphere MQ queue manager:

```
STRMQM MQMNAME(QMName)
```

Then run the default MQSC script by typing the following command:

```
STRMQMMQSC SRCMBR(AMTSDFTS) SRCFILE(QMQMAMI/AMTMQSC) MQMNAME(QMName)
```

For UNIX and Windows, first start the local queue manager by typing the following at a command line:

```
strmqm {QMName}
```

where `{QMName}` is the name of your WebSphere MQ queue manager.

Then run the default MQSC script by typing one of the following:

```
  runmqsc {QMName} < {Location}/amtsdfts.tst    (UNIX)

  runmqsc {QMName} < {Location}\amtsdfts.tst    (Windows)
```

where `{QMName}` is the name of your WebSphere MQ queue manager and `{Location}` is the location of the `amtsdfts.tst` file.

For z/OS, start the local queue manager, then use the CSQUTIL program to run the default MQSC script:

```
//COMMAND EXEC PGM=CSQUTIL,PARM='QMGR'
//STEPLIB  DD DSN=hlq.SCSQAUTH,DISP=SHR
//         DD DSN=hlq.SQSCANLE,DISP=SHR
//AMTSDFTS DD DSN=hlq.SCSQPROC(AMTSDFTS),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
COMMAND DDNAME(AMTSDFTS)
/*
```

where `hlq` is the high level qualifier of your WebSphere MQ installation, and `QMGR` is your queue manager name.

The sample programs

Sample programs are provided to illustrate the use of the Application Messaging Interface.

We recommend that you run one or more of the sample programs to verify that you have installed the Application Messaging Interface correctly.

For the z/OS platform, see "Sample programs for z/OS" on page 483.

Sample programs for iSeries, UNIX, and Windows

There are ten basic sample programs for iSeries, UNIX, and Windows platforms, performing approximately the same function in C, C++, and Java. Consult the source code to find out how the programs achieve this functionality. The C samples are provided for both the high-level interface and the object interface.

Table 5. The sample programs for iSeries, UNIX, and Windows platforms

Description	C high-level	C object- level	C++	Java
A sample that sends a datagram message, expecting no reply.	amtshsnd	amtsosnd	SendForget	SendForget
A sample that receives a message, with no selection.	amtshrcv	amtsorcv	Receiver	Receiver
A sample that sends a request and receives a reply to this request (a simple client program).	amtshclt	amtsoclt	Client	Client
A sample that receives requests and sends replies to these requests (a simple server program).	amtshsvr	amtsosvr	Server	Server
A sample that periodically publishes information on the weather.	amtshpub	amtsopub	Publisher	Publisher
A sample that subscribes to information on the weather, and receives publications based on this subscription.	amtshsub	amtsosub	Subscriber	Subscriber
A sample that sends messages using simulated group support.	-	amtsosgs	-	-
A sample that receives messages using simulated group support.	-	amtsosgr	-	-
A sample that performs a file transfer send on a user supplied text file.	amtshfsn	amtsofsn	SendFile	SendFile
A sample that performs a file transfer receive on a user supplied text file.	amtshfrc	amtsofrc	RcvFile	RcvFile
A sample policy handler library that inserts a CICS (MQCIH) or IMS (MQIIH) header at the start of each message that is sent, or that performs archiving by sending each message that is sent to a separate archive queue.	-	amtsphlr	-	-

To find the source code and the executables for the samples, see "Directory structure" on page 445 (AIX), page 449 (HP-UX), page 453 (iSeries), page 458 (Solaris), and page 462 (Windows).

Running the iSeries, UNIX, and Windows sample programs

Before you can run the sample programs on iSeries, UNIX, or Windows platforms, you must make a number of preparations. These are described in the following sections.

WebSphere MQ objects

The sample programs require some WebSphere MQ objects to be defined. To do this, use a WebSphere MQ MQSC file, `amtsamp.tst`, which is shipped with the samples.

For iSeries, start the local queue manager by typing the following on the command line:

```
STRMQM MQMNAME(QMName)
```

where `QMName` is the name of your WebSphere MQ queue manager.

Then run the default MQSC script by typing the following command:

```
STRMQMMQSC SRCMBR(AMTSAMP) SRCFILE(QMQMAMI/AMTMQSC) MQMNAME(QMName)
```

For UNIX or Windows, start the local queue manager by typing the following at a command line:

```
strmqm {QMName}
```

where `{QMName}` is the name of your WebSphere MQ queue manager.

Then run the sample MQSC script by typing one of the following:

```
runmqsc {QMName} < {Location}/amtsamp.tst    (UNIX)

runmqsc {QMName} < {Location}\amtsamp.tst    (Windows)
```

where `{QMName}` is the name of your WebSphere MQ queue manager and `{Location}` is the location of the `amtsamp.tst` file.

Repository and host files

Copy the sample repository file, `amt.xml`, into the default location for your platform (see “Local host and repository files (iSeries, UNIX, and Windows)” on page 468).

Modify the host file so that your WebSphere MQ queue manager name, {QMName}, is known as `defaultConnection`.

Running the publish/subscribe samples

To run the AMI publish/subscribe samples, you need access to a WebSphere MQ broker. This can be either WebSphere MQ Publish/Subscribe or WebSphere MQ Integrator Version 2. You can issue publish/subscribe requests locally or remotely to either broker. If the platform on which the requesting application runs does not support a WebSphere MQ broker, publish/subscribe requests can only be issued remotely. In this situation, you must set up the appropriate WebSphere MQ channels, and ensure that the remote queue manager and channels are started.

WebSphere MQ Publish/Subscribe broker: To run the publish/subscribe samples with WebSphere MQ Publish/Subscribe broker, you must start the broker. Type the following at a command line:

```
strmqbrk -m {QMName}
```

where `{QMName}` is the name of your WebSphere MQ queue manager.

WebSphere MQ Integrator Version 2: To run the publish/subscribe samples with WebSphere MQ Integrator Version 2 you need to do the following:

1. Start the broker and the Configuration Manager. Type the following at a command line, where `{BrokerName}` is the name of your WebSphere MQ Integrator Version 2 broker:
   ```
   mqsistart  {BrokerName}
   mqsistart  ConfigMgr
   ```
2. Use the Control Center to create a simple message flow. This should consist of an MQInput node with the Output terminal connected to the input terminal of a Publication node.
3. In the properties for the MQInput node, select the "Basic Properties" pane and set the Queue Name to:
   ```
   SYSTEM.BROKER.DEFAULT.STREAM
   ```

   For iSeries only:
   1. In the properties for the MQInput node, select the "Advanced" properties pane and check the Convert check box.
   2. Set the Convert Encoding and Convert Coded Character Set ID to the native values used by WebSphere MQ for the platform where the broker is running.

      For example, on Windows NT broker, set the Convert Encoding to 546 (that is, the MQENC_NATIVE value), and set the Convert Coded Character Set ID to 850.
4. Add the new message flow to the execution group for your Broker and deploy it.

For further details, refer to the *WebSphere MQ Integrator Version 2.0 Programming Guide*.

Setting the runtime environment

Before you run the AMI samples, make sure that you have set up the runtime environment. See "Setting the runtime environment" on page 444 (AIX), page 448 (HP-UX), page 452 (iSeries), page 457 (Solaris), and page 461 (Windows).

Running the C and C++ samples

You can run a C or C++ sample program by typing the name of its executable at a command line. For example:

```
amtsosnd
```

will run the "Send and forget" sample written using the C object interface.

Running the Java samples

The AMI samples for Java are in a package called:

```
com.ibm.mq.amt.samples
```

To invoke them, you need to specify the name of the sample plus its package name. For example, to run the "Send and forget" sample, use:

```
java com.ibm.mq.amt.samples.SendForget
```

Running the policy handler library sample

The sample policy handler library (AMT.SAMPLE.HANDLER) archives each message when it is sent using either a sender or distribution list, by putting a copy of the message to an additional WebSphere MQ queue (AMT.SAMPLE.POLICY.HANDLER.QUEUE) on the local queue manager.

To run the policy handler, the WebSphere MQ queue AMT.SAMPLE.POLICY.HANDLER.QUEUE must exist. To create this queue, you can run the amtsamp.tst script file when prompted during installation, or you can run the sample MQSC script after installation (see “WebSphere MQ objects” on page 481).

Also, the policy handler must be defined in the repository and referenced in the policy specified with the send function by an application. The sample repository file (amt.xml) that is provided with the AMI includes a policy handler definition named AMT.SAMPLE.HANDLER for this library. It also includes a policy definition named AMT.SAMPLE.POLHDLR.POLICY that includes a reference to AMT.SAMPLE.HANDLER. See “Repository and host files” on page 481.

You can use the policy AMT.SAMPLE.POLHDLR.POLICY to exercise the sample policy handler by recompiling amtshclt.c or amtsoclt.c with the AMT_RUN_HANDLERS preprocessor directive specified (that is, using the `-D` option with the `cl` command). This causes the AMT.SAMPLE.POLHDLR.POLICY policy name to be used instead of the one that is normally used.

For a more detailed description, see “The AMI policy handler sample program (amtsphlr)” on page 487.

Running the sample programs (iSeries)

Executable sample programs are provided in the QMQMAMI library. To run C and C++ samples on iSeries, use CALL, followed by the name of the executable. For example, to run the “Send and Forget” sample written using the C object interface, enter:

```
CALL AMTSOSND
```

Sample programs for z/OS

There are ten basic sample programs in C for the z/OS platform, and a matching set in COBOL that perform approximately the same function. Consult the source code to find out how the programs achieve this functionality. The samples are provided for both the high-level interface and the object-level interface in most cases.

There is also a C header file `amts39sp` that implements environment-specific I/O functions for CICS and IMS. This header file is not required to build the samples for Batch.

Table 6. The sample programs for z/OS (‘batch’ includes RRS-batch)

Description	**C High level**	**C Object level**	**COBOL High level**	**COBOL Object level**
A sample that sends a datagram message, expecting no reply.	AMTSHSND	AMTSOSND	AMTVHSND	AMTVOSND
A sample that receives a message, with no selection.	AMTSHRCV	AMTSORCV	AMTVHRCV	AMTVORCV
A sample that sends a request and receives a reply to this request (a simple client program).	AMTSHCLT	AMTSOCLT	AMTVHCLT	AMTVOCLT
A sample that receives requests and sends replies to these requests (a simple server program).	AMTSHSVR	AMTSOSVR	AMTVHSVR	AMTVOSVR
A sample that periodically publishes information on the weather.	AMTSHPUB	AMTSOPUB	AMTVHPUB	AMTVOPUB

Table 6. The sample programs for z/OS (‘batch’ includes RRS-batch) (continued)

Description	C High level	C Object level	COBOL High level	COBOL Object level
A sample that subscribes to information on the weather, and receives publications based on this subscription.	AMTSHSUB	AMTSOSUB	AMTVHSUB	AMTVOSUB
A sample that sends simulated group messages. This uses object-level calls only.	Not applicable	AMTSOSGS	Not applicable	AMTVOSGS
A sample that receives simulated group messages. This uses object-level calls only.	Not applicable	AMTSOSGR	Not applicable	AMTVOSGR
A sample that performs a file transfer send on a user-supplied text file. Not for use under CICS.	AMTSHFSN	AMTSOFSN	AMTVHFSN	AMTVOFSN
A sample that performs a file transfer receive on a user-supplied text file. Not for use under CICS.	AMTSHFRC	AMTSOFRC	AMTVHFRC	AMTVOFRC
A sample policy handler library that inserts a message header or performs archiving. See “The AMI policy handler sample program (amtsphlr)” on page 487.	AMTSPHLR	-	-	-

To find the source code for the samples, see “Directory structure (z/OS)” on page 465.

Running the sample programs (z/OS)

Before you can run the sample programs on the z/OS platform, there are a number of actions to take.

Building the sample programs

The samples for z/OS are provided as source code only, so you must build them before you can run them. See “Building C applications” on page 31 and “COBOL applications on z/OS” on page 260.

WebSphere MQ objects

The sample programs require some WebSphere MQ objects to be defined. This can be done with a WebSphere MQ MQSC file, `AMTSAMP`, which is shipped with the samples.

First start the local queue manager, as described in the *WebSphere MQ for z/OS System Administration Guide*. If you are using the CICS environment, ensure that the WebSphere MQ CICS adapter is set up and the CICS region is connected to the queue manager.

Then run the sample MQSC script AMTSAMP (located in the `hlq.SCSQPROC` dataset) using the WebSphere MQ utility program CSQUTIL. Following is a JCL fragment to help you run the utility:

```
//COMMAND EXEC PGM=CSQUTIL,PARM='QMGR'
//STEPLIB  DD DSN=hlq.SCSQAUTH,DISP=SHR
//         DD DSN=hlq.SQSCANLE,DISP=SHR
//AMTSAMP  DD DSN=hlq.SCSQPROC(AMTSAMP),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
COMMAND DDNAME(AMTSAMP)
/*
```

where `hlq` is the high level qualifier of your WebSphere MQ installation, and `OMGR` is your queue manager name.

Repository and host files

The sample repository `AMT` (located in `hlq.SCSQPROC`) is appropriate for use with all the sample programs (though many of the samples will work correctly without a repository). If you wish to use the repository file, ensure that the sample program has access to it, as described in “Local host and repository files (z/OS)” on page 470.

For batch, RRS-batch, and IMS programs (not CICS), copy the sample host file `AMTHOST` (UTF-8) or `AMTHOST2` (EBCDIC) from `hlq.SCSQPROC` to another location, and modify it so that your WebSphere MQ queue manager name is `defaultConnection`. Ensure that the sample program has access to the host file, using DD statements as described in “Local host and repository files (z/OS)” on page 470.

Running the publish/subscribe samples

To use the publish/subscribe samples, you need access to a WebSphere MQ Publish/Subscribe broker. Because this is not available on z/OS, you must have a WebSphere MQ queue manager and publish/subscribe broker running on another platform. In this situation, you must set up the appropriate WebSphere MQ channels, and ensure that the remote queue manager and channels are started.

WebSphere MQ Publish/Subscribe broker: To run the publish/subscribe samples with WebSphere MQ Publish/Subscribe broker, you must start the broker. Type the following at a command line:

```
strmqbrk -m {QMName}
```

where `{QMName}` is the name of your WebSphere MQ queue manager.

WebSphere MQ Integrator Version 2: To run the publish/subscribe samples with WebSphere MQ Integrator Version 2 you need to do the following:

1. Start the broker and the Configuration Manager. Type the following at a command line, where `{BrokerName}` is the name of your WebSphere MQ Integrator Version 2 broker:
   ```
   mqsistart  {BrokerName}
   mqsistart  ConfigMgr
   ```
2. Use the Control Centre to create a simple message flow. This should consist of an MQInput node with the Output terminal connected to the input terminal of a Publication node.
3. In the properties for the MQInput node, select the “Basic Properties” pane and set the Queue Name to:
   ```
   SYSTEM.BROKER.DEFAULT.STREAM
   ```
4. In the properties for the MQInput node, select the “Advanced” properties pane and check the Convert check-box.
5. Set the Convert Encoding and Convert Coded Character Set ID to the native values used by WebSphere MQ for the platform where the broker is running.

   For example, on Windows NT broker, set the Convert Encoding to 546 (i.e., the MQENC_NATIVE value) and the Convert Coded Character Set ID to 850.
6. Add the new message flow to the execution group for your Broker and deploy it.

For further details, refer to the *WebSphere MQ Integrator Version 2.0 Programming Guide*.

Setting the runtime environment

Make sure your environment has been set to pick up the AMI runtime binary files, as described in “Setting the runtime environment” on page 465.

File name input for the file transfer samples

There are 3 ways in z/OS to specify the file name for the file transfer samples:

1. Use single quotes.
   ```
   // PARM='MYTEST.FILE'
   ```
   It will then obey z/OS rules and be prefixed with the user’s RACF® ID as the high level qualifier.
2. Supply a fully qualified filename using double quotes, with an extra outer pair to contain the parm data.
   ```
   // PARM= '"userId.MYTEST.FILE"'
   ```
3. Supply the keywords `DD:FILE` in the parms where FILE is the DD NAME.
   ```
   // PARM='DD:MYFILE'
   .
   .
   // MYFILE    DD  DSNAME=userId.MYTEST.FILE,DISP=SHR
   ```

Each method resolves to `userId.MYTEST.FILE`.

Running the batch samples

You can run batch sample programs by constructing a piece of JCL to run the program, and submitting that JCL from ISPF. The batch samples can also be used as RRS-batch sample programs.

Running the CICS samples

Ensure that the CICS DFHRPL includes the load library containing the sample, as well as the AMI library. Define the sample program to CICS, as well as a transaction to run the program. Finally, ensure that the AMI library, sample program and sample transaction are installed in your CICS region. Type the transaction name into a CICS console to run the sample.

Running the IMS samples

Ensure that the IMS message processing region JCL includes the load library that contains the sample, as well as the AMI library. Define the sample program and transaction name to IMS. Type the transaction name into an IMS console to run the sample.

Running the z/OS sample policy handler

The z/OS sample policy handler, AMTSPHLR, contains the same code as the sample policy handler that is supplied for distributed platforms. For a functional description, see “The AMI policy handler sample program (amtsphlr)” on page 487. The supplied z/OS sample is designed for use only with AMI applications running as batch jobs. However, policy handler functions are supported in all z/OS environments that support AMI, where different methods for input and output may be necessary (for example, the supplied sample uses the `printf` command).

Preparing the z/OS sample policy handler: The sample for z/OS is supplied as source code only, so you must build it before use. A policy handler for z/OS is built in a similar fashion to a z/OS AMI C application program. The specific requirements are:

Compile: Make sure that a DD statement for the dataset `hlq.SCSQC370` is added to the C compiler’s SYSLIB concatenation. The C compiler options must include DLL, LONGNAME, and REENTRANT.

Pre-link: A DD statement for the sidedeck member, `hlq.SCSQDEFS(AMTBD10)`, must be specified in the pre-link step SYSIN concatenation after the application object code member.

The policy handler code contains its own exported functions, so that as a by-product, the pre-link produces an output sidedeck. Therefore, a DD statement for SYSDEFSD must be supplied, although this may be specified as DUMMY, because this dataset is not required.

Link-edit: The sample program invokes WebSphere MQ functions. Therefore, a DD statement for `hlq.SCSQLOAD` must be included in the SYSLIB concatenation. Also, an INCLUDE statement for the WebSphere MQ batch stub CSQBSTUB must be added to SYSIN.

Running the z/OS sample policy handler: The defined policy handler functions are invoked during an AMI session if the AMI policy specifies an appropriate handler. The supplied sample repository defined in AMT.XML includes a policy definition AMT.SAMPLE.POLHDLR.POLICY, which defines a handler AMT.SAMPLE.HANDLER. AMT.SAMPLE.HANDLER defines AMTSPHLR as the policy handler library (DLL) name. Therefore, to use the sample policy handler, use the supplied sample repository and specify AMT.SAMPLE.POLHDLR.POLICY as the policy for the AMI session in the AMI application program.

The AMI policy handler sample program (amtsphlr)

The AMI Policy Handler sample program, amtsphlr, is implemented using the following files:

- `amtsphlh.h` : C header file for policy handler sample
- `amtsphlr.c` : C source for policy handler sample
- `amtsphlr.exp` : AIX and HP-UX C export file for policy handler sample
- `amtsphlr.def` : Windows C definition file for policy handler sample

For iSeries, the AMI Policy Handler sample program is implemented using the following files:

- `amtsphlh` in QMQMAMI/H : C header file for policy handler sample
- `amtsphlr` in QMQMAMI/QCSRC : C source for policy handler sample
- `amtsphlr` in QMQMAMI/QSRVSRC : AIX and HP-UX C export file for policy handler sample

This sample program illustrates the following policy handler operations:

- Creation. The creation of a policy handler context and the return of a context handle.
- Initialization. The initialization of the policy handler and invocation point registration.

  The sample policy handler registers functions for the following invocation points:

  ```
  AMINV_PRE_MQCONNX
  AMINV_PRE_MQOPEN
  AMINV_PRE_MQPUT
  ```
- Deletion. The deletion of the policy handler context.
- Invocation. The invocation of a policy handler to perform a simple invocation print function.

- Auditing. The invocation of a policy handler with policy initialization parameters used as a directive to log each message on a separate audit queue when it is sent.
- Header insertion. The invocation of a policy handler with a service custom parameter used as directive to insert an additional header at the start of the message. This sample program illustrates WebSphere MQ CICS (MQCIH) and IMS (MQIIH) header insertion, with policy invocation parameters used to provide the CICS or IMS program name.

These functions illustrate the use of the **amLibTraceText** AMI callback function to perform tracing when requested.

A definition for the sample policy handler is provided in the sample repository file `amt.xml` that is provided with AMI. This specifies the policy handler AMT.SAMPLE.HANDLER with library name amtsphlr. The corresponding library file must be located in the AMI handlers directory for the platform concerned. The name of the corresponding library file requires the prefix `lib` on AIX, HP-UX and Sun Solaris, and requires the appropriate suffix for the platform concerned:

`.a`	AIX (non-threaded)
`_r.a`	AIX (threaded)
`_r`	iSeries (threaded)
`_r.sl`	HP-UX
`.so`	Solaris
`.dll`	Windows

The policy handler name AMT.SAMPLE.HANDLER is also specified in the list of handlers for the policy AMT.SAMPLE.POLHDLR.POLICY. To invoke this policy handler, an application must create a policy object with the name AMT.SAMPLE.POLHDLR.POLICY and specify this as a parameter on an AMI open or send function. To invoke this policy handler using the `amtsoclt.c` or `amtshclt.c` sample programs, you must recompile this program using the -D AMT_RUN_HANDLERS compiler option (to define the symbol AMT_RUN_HANDLERS).

For details about how to run the sample policy handler, see “Running the policy handler library sample” on page 482 and “Running the z/OS sample policy handler” on page 486.

Required entry points

The sample policy handler library implements and exports the three required entry points as follows:

amPhlCreate

This is called on creation of the first policy to include this policy handler in its list of handlers. It allocates the memory required for its own instance data, and returns a context handle that will be passed back to the policy handler on each subsequent call.

The context handle enables the policy handler to manage and access its own memory context information across invocations. The value returned for the context handle must be non-NULL. A policy handler context has session scope, that is a separate **amPhlCreate** for the same policy handler can occur for each AMI session.

The context handle can itself be used to hold the context pointer if its size is suitable. Otherwise, on platforms where this is not possible, the pointer must be stored in global memory. Note that when the pointer is stored in global memory, if multiple sessions reference the same policy handler, the context handle can serve as an index to a pointer table in global memory. However, this is not illustrated in this sample program.

amPhlInitialize

This is called immediately after **amPhlCreate** to enable the policy handler to call back into the AMI and register the set of invocation points for which it wishes to be called and specify their function address values. This sample program registers the following functions and invocation points:

Function Name	Invocation point
InvocationPrint	AMINV_PRE_MQCONNX
InvocationPrint	AMINV_PRE_MQOPEN
AddHeader	AMINV_PRE_MQPUT
Audit	AMINV_POST_MQPUT

amPhlDelete

This is called when the session is deleted to enable the policy handler to free any memory it allocated during **amPhlCreate** processing.

Invocation point functions

The sample policy handler also implements the following functions that it registers with the required invocation points during **amPhlInitialize** processing, as listed earlier. These implement the real function provided by the policy handler.

InvocationPrint

This function prints a message that identifies the invocation point (that is, in this example, either AMINV_PRE_MQCONNX or AMINV_PRE_MQOPEN).

Audit This function is invoked after a successful MQPUT. If the handler initialization parameters specified AUDIT, this function logs a copy of the same message to a separate WebSphere MQ audit queue, that is:

AMT.SAMPLE.POLICY.HANDLER.QUEUE on the local queue manager to which the AMI session is connected.

AddHeader

This function is invoked immediately before an MQPUT.

If the custom parameters for the sender service specify CICS, this function constructs a WebSphere MQ MQCIH (CICS) message header and uses the AMI call back function **amxMsgAssemble** to add this at the start of the message. The policy invocation parameters are used to specify the CICS program name.

If the custom parameters for the sender service specify IMS, this function constructs a WebSphere MQ MQIIH (IMS) message header and this uses the AMI call back function **amxMsgAssemble** to add this at the start of the message. The policy invocation parameters are used to specify the IMS program name.

The function returns AMPH_CONTINUE in the control flags, which causes the AMI to issue the MQPUT to send the modified message.

The sample programs

If you use a policy handler to add a CICS header, this enables your AMI application to run CICS Distributed Program Link (DPL) programs on a CICS Transaction Server using WebSphere MQ and the WebSphere MQ-CICS bridge as the communication mechanism.

If you use a policy handler to add an IMS header, this enables your AMI application to run IMS/ESA transactions on an IMS Transaction Server using WebSphere MQ and the WebSphere MQ-IMS bridge as the communication mechanism.

Chapter 18. Defining services, policies, and policy handlers

Definitions of services, policies, and policy handlers created by a system administrator are held in a *repository*. The Application Messaging Interface provides a tool to enable the administrator to set up new services, policies, and policy handlers, and to specify their attributes (see "The AMI Administration Tool" on page 475).

This chapter contains:
- "Services, policies, and policy handlers"
- "Service definitions" on page 494
- "Policy definitions" on page 497
- "Policy handler definitions" on page 504

Services, policies, and policy handlers

A repository file contains definitions for *policies*, *policy handlers*, and *services*. A service is the generic name for any object to which a send or receive request can be issued, that is:
- Sender
- Receiver
- Distribution list
- Subscriber
- Publisher

Sender and receiver definitions are represented in the repository by a single definition called a *service point*.

Policies, policy handlers, and services other than distribution lists, can be created with or without a corresponding repository definition; distribution lists can be created only with a corresponding repository definition.

To create a service, policy, or policy handler using the repository, the repository must contain a definition of the appropriate type with a name that matches the name specified by the application. To create a sender object named 'DEBITS' (using **amSesCreateSender** in C, for example) the repository must have a service point definition named 'DEBITS'.

Policies, policy handlers, and services created with a repository have their contents initialized from the named repository definition.

If the repository does not contain a matching name, a warning is issued (such as AMRC_POLICY_NOT_IN_REPOS). The service or policy is then created without using the repository (unless it is a distribution list).

Policies and services created without a repository (either for the previous reason, or because the repository is not used), have their contents initialized from one of the system provided definitions (see "System provided definitions" on page 492).

Definition names in the repository must not start with the characters 'AMT' or 'SYSTEM'.

System provided definitions

The AMI provides a set of definitions for creating services, policies, and policy handlers without reference to a repository.

Table 7. System provided definitions

Definition	Description
AMT.SYSTEM.POLICY	This provides a policy definition with the defaults specified in "Policy definitions" on page 497, except that Wait Interval Read Only is not selected in the Receive attributes.
AMT.SYSTEM.SYNCPOINT.POLICY	This provides a policy definition the same as AMT.SYSTEM.POLICY, except that Syncpoint is selected in the General attributes.
AMT.SYSTEM.SENDER	This provides a sender definition with the defaults specified in "Service definitions" on page 494, with the Queue Name the same as the Sender object.
AMT.SYSTEM.RESPONSE.SENDER	This provides a sender definition the same as AMT.SYSTEM.SENDER, except that Definition Type, Queue Name and Queue Manager Name are set to 'Undefined' (that is, set when used).
AMT.SYSTEM.RECEIVER	This provides a receiver definition the same as AMT.SYSTEM.SENDER.
AMT.SYSTEM.SUBSCRIBER	This provides a subscriber definition in which the Sender Service has the same name as the Subscriber object, and the Receiver Service has the same name with the suffix '.RECEIVER'.
AMT.SYSTEM.PUBLISHER	This provides a publisher definition in which the Broker Service has the same name as the Publisher object.

System default objects

A set of system default objects is created at session creation time. This removes the overhead of creating the objects from applications using these defaults. The system default objects are available for use from the high-level and object-level interfaces in C. They cannot be accessed using C++ or Java (these languages can use the built-in definitions to create an equivalent set of objects if required).

The default objects are created using the system provided definitions, as shown in the following table.

Table 8. System default objects

Default object	Definition
SYSTEM.DEFAULT.POLICY	AMT.SYSTEM.POLICY
SYSTEM.DEFAULT.SYNCPOINT.POLICY	AMT.SYSTEM.SYNCPOINT.POLICY
SYSTEM.DEFAULT.SENDER	AMT.SYSTEM.SENDER
SYSTEM.DEFAULT.RESPONSE.SENDER	AMT.SYSTEM.RESPONSE.SENDER
SYSTEM.DEFAULT.RECEIVER	AMT.SYSTEM.RECEIVER
SYSTEM.DEFAULT.SUBSCRIBER	AMT.SYSTEM.SUBSCRIBER
SYSTEM.DEFAULT.PUBLISHER	AMT.SYSTEM.PUBLISHER

Table 8. System default objects (continued)

Default object	Definition
SYSTEM.DEFAULT.SEND.MESSAGE	N/A
SYSTEM.DEFAULT.RECEIVE.MESSAGE	N/A

The default objects can be used explicitly using the AMI constants (see Appendix B, “Constants and structures” on page 561), or used to provide defaults if a particular parameter is omitted (by specifying NULL, for instance).

Handle synonyms are also provided for these objects, for use from the object interface (see Appendix B, “Constants and structures” on page 561). Note that the first parameter on a call must be a real handle; you cannot use a synonym handle in this case.

Service definitions

This section gives the service definitions for:

- service point (sender/receiver)
- distribution list
- subscriber
- publisher

Service point (sender/receiver)

Table 9. Service point (sender/receiver)

Attribute	**Comments**
Name	Mandatory name, specified on AMI calls. 1
Queue Name	Mandatory name of the queue representing the service that messages are sent to or received from. 2
Queue Manager Name	Name of the queue manager that owns Queue Name. If blank, the local queue manager name is used. 2
Model Queue Name	Name of a model queue definition used to create a dynamic queue (normally a Reply Service to receive response messages). Required if the Definition Type is 'Dynamic'. 2
Dynamic Queue Prefix	Name of a prefix used when creating a dynamic queue from Model Queue Name. Required if the Definition Type is 'Dynamic'. If the last non-blank character in positions 1 to 33 of the prefix is `'*'`, the `'*'` is replaced by a string that guarantees that the name generated is unique. 2
Definition Type	Defines how the AMI obtains the queue name for the service point. If set to 'Predefined' (the default), the Queue Name and Queue Manager Name as specified earlier are used. If set to 'Dynamic', the Model Queue Name and Dynamic Queue Prefix are used to create a dynamic queue.
Service Type	Defines the RF header (if any) that is sent with the message data, and the parameters within the header. Set to 'Native' for a native WebSphere MQ service (default). Set to 'MQSeries Integrator V1' for MQSeries Integrator Version 1 (adds the OPT_APP_GROUP and OPT_MSG_TYPE fields to the MQRFH header). Set to 'RF Header V1' for MQSeries Publish/Subscribe applications. Set to 'WebSphere MQ Integrator V2' to use the appropriate publish and subscribe policy options when sending publish, subscribe and unsubscribe requests to the WebSphere MQ Integrator Version 2 broker. The AMI will insert each of the (non-blank) default MCD values defined for the service point into any message being sent using this service point. If Service Type is set to 'RF Header V2', a Version 2 RF Header will be used when applicable but the WebSphere MQ Integrator V2 specific policy properties (Default MCD Domain, Default MCD Set, Default MCD Type, Default MCD Format, Delivery Persistence and Subscription Point) are not added to the message.
Default Format	Optional format name to insert in the MQMD, if a format value of FMT_NONE is set in the message object. Also used as the MsgType when the service is a WebSphere MQ Integrator Version 1 broker, if AMFMT_NONE is set in the message object and the MsgType has not been added explicitly (using **amMsgAddElement** or equivalent). 3
Default MCD Domain	Defines the default message service domain value. This is added to any message being sent using this service point if the Service Type is 'WebSphere MQ Integrator V2', the value of this field is non-blank and a message service domain element has not been explicitly added to the message by the application. 1 4

Table 9. Service point (sender/receiver) (continued)

Attribute	Comments
Default MCD Set	Defines the default message set value. This is added to any message being sent using this service point if the Service Type is 'WebSphere MQ Integrator V2', the value of this field is non-blank, and a message set element has not been explicitly added to the message by the application. [1] [4]
Default MCD Type	Defines the default message type value. This is added to any message being sent using this service point if the Service Type is 'WebSphere MQ Integrator V2', the value of this field is non-blank, and a message type element has not been explicitly added to the message by the application. [1] [4]
Default MCD Format	Defines the default message format value. This is added to any message being sent using this service point if the Service Type is 'WebSphere MQ Integrator V2', the value of this field is non-blank, and a message format element has not been explicitly added to the message by the application. [1] [4]
CCSID	Coded character set identifier of the destination application. Can be read by sending applications in order to prepare a message in the correct CCSID for the destination. It is not used by the AMI. Leave blank if the CCSID is unknown (the default), or set to the CCSID number. [5]
Encoding	Integer encoding of the destination application. Can be read by sending applications in order to prepare a message in the correct encoding for the destination. It is not used by the AMI. Set to 'Unspecified' (the default), 'Reversed', 'Normal', 'Reversed With 390 Floating Point', or 'Normal With 390 Floating Point'.
Simulated Group Support	Select to enable the sending and receiving of messages that form part of a message group to or from a target WebSphere MQ queue manager that does not provide native support for groups (see "Sending group messages" on page 27).
Custom Parameters	Free-format text parameter string that is associated with the service point and passed to the policy handler library on each invocation to provide service-specific information. The implementor of the policy handler defines the meaning and syntax of the string. The string is specific to the policy handler library, and the policy handler library parses and interprets the string. A typical use of this field might be to further customize the service-type as CICS or IMS, where the policy handler library is responsible for inserting and removing a MQCIH or MQIIH header at the start of the message.

Notes:

[1] The name is a maximum of 256 characters, and can contain the following characters: `A-Z`, `a-z`, `0-9`, `'.'`, `'/'`, `'_'` and `'%'`.

[2] The name is a maximum of 48 characters, and can contain the following characters: `A-Z`, `a-z`, `0-9`, `'.'`, `'/'`, `'_'` and `'%'`.

[3] The name is a maximum of 8 characters, and can contain any character from a single byte character set (it is recommended that the characters are restricted to `A-Z`, `0-9`).

[4] This attribute is applicable only for Service Type 'WebSphere MQ Integrator V2' and is ignored for other Service Type settings.

[5] The name is a maximum of 6 characters, and can contain any numeric character.

Distribution list

Table 10. Distribution list

Attribute	Comments
Name	Mandatory name, specified on AMI calls. **1**
Available Service Points	List of service points that make up the distribution list. They must be valid service point names.
Note: **1** The name is a maximum of 256 characters, and can contain the following characters: `A-Z`, `a-z`, `0-9`, `'.'`, `'/'`, `'_'` and `'%'`.	

Subscriber

Table 11. Subscriber

Attribute	Comments
Name	Mandatory name, specified on AMI calls. **1**
Sender Service	The name of the sender service that defines the publish/subscribe broker. It must be a valid service point name.
Receiver Service	The name of the receiver service that defines where publication messages are to be sent. It must be a valid service point name.
Note: **1** The name is a maximum of 256 characters, and can contain the following characters: `A-Z`, `a-z`, `0-9`, `'.'`, `'/'`, `'_'` and `'%'`.	

Publisher

Table 12. Publisher

Attribute	Comments
Name	Mandatory name, specified on AMI calls. **1**
Sender Service	The name of a sender service that defines the publish/subscribe broker. It must be a valid service point name.
Note: **1** The name is a maximum of 256 characters, and can contain the following characters: `A-Z`, `a-z`, `0-9`, `'.'`, `'/'`, `'_'` and `'%'`.	

Policy definitions

This section describes the policy definitions for the following attributes:
- initialization
- general
- send
- receive
- subscribe
- publish
- handler

Initialization attributes

Table 13. Initialization attributes

Attribute	Comments
Name	Mandatory policy name, specified on AMI calls. 1
Connection Name	If Connection Mode is set to ‘Real’, Connection Name is the name of the queue manager the application will connect to. If blank, the default local queue manager is used. If Connection Mode is ‘Logical’, the Connection Name attribute is required and is the name of the logical connection used with the local host file to generate the queue manager to which connection is made. 2
Connection Mode	If Connection Mode is set to ‘Real’ (the default), Connection Name is used as the queue manager name for connection. If Connection Mode is set to ‘Logical’, Connection Name is used as a key to the host file on the system where the application is running that maps Connection Name to a queue manager name. This allows applications running on different systems in the network to use the same repository (connection name) to connect to different local queue managers.
Connection Type	If Connection Type is set to ‘Auto’ (the default), the application automatically detects if it should connect directly, or as a client. If Connection Type is ‘Client’, the application connects as a client. If Connection Type is ‘Server’, the application connects directly to the queue manager. 3
Trusted Option	If set to ‘Normal’ (the default), no fastpath is used. If set to ‘Trusted’, the application can use fastpath facilities that might compromise integrity. This option is only supported on Windows.
Client Channel Name	For a WebSphere MQ client connection, the name of the server-connection channel. Can be used instead of the MQSERVER environment variable on the WebSphere MQ client with the TCP/IP transport type.
Client TCP Server Address	For a WebSphere MQ client connection, the TCP/IP host name (and optional port) of the WebSphere MQ server. Can be used instead of the MQSERVER environment variable on the WebSphere MQ client with the TCP/IP transport type.

Notes:

1 The name is a maximum of 256 characters, and can contain the following characters: `A-Z`, `a-z`, `0-9`, `'.'`, `'/'`, `'_' and '%'`.

2 The name is a maximum of 48 characters, and can contain the following characters: `A-Z`, `a-z`, `0-9`, `'.'`, `'/'`, `'_' and '%'`.

3 The Connection Type that is established on the first session open persists for the entire process. You cannot change this by opening a subsequent session using a policy with a different Connection Type.

General attributes

Table 14. General attributes

Attribute	Comments
Message Context	Defines how the message context is set in messages sent by the application. The default is 'Set By Queue Manager' (the queue manager sets the context). If set to 'Pass Identity', the identity of the request message is passed to any output messages. If set to 'Pass All', all the context of the request message is passed to any output messages. If set to 'No Context', no context is passed.
Syncpoint	If selected, the send or receive is part of a unit of work (default is 'not selected').

Send attributes

Table 15. Send attributes

Attribute	Values	Default	Comments
Implicit Open	Selected Not selected	Selected	When selected, the queue is opened implicitly (must be selected for the C and COBOL high-level interfaces). **1**
Leave Queue Open	Selected Not selected	Selected	When selected, a queue that was implicitly opened will be left open. **1**
Priority	0-9 As Transport	As Transport	The priority set in the message, where 0 is the lowest priority and 9 is the highest. When set to As Transport, the value from the queue definition is used. You must deselect As Transport before you can set a priority value.
Persistence	Yes No As Transport	As Transport	The persistence set in the message, where Yes is persistent and No is not persistent. When set to As Transport, the value from the underlying queue definition is used.
Expiry Interval	0-999999999 Unlimited	Unlimited	A period of time (in tenths of a second) after which the message will not be delivered.
Retry Count	0-999999999	0	The number of times a send will be retried if the return code gives a temporary error. Retry is attempted under the following conditions: Queue full, Queue disabled for put, Queue in use.
Retry Interval	0-999999999	1000	The interval (in milliseconds) between each retry.
New Correl Id	Selected Not selected	Not selected	When selected, each message is sent with a new Correl Id (except for response messages, where this is set to the Message Id or Correl Id of the request message).
Response Correl Id	Message Id Correl Id	Message Id	The Id set in the Correl Id of a response or report message. This is set to either the Message Id or the Correl Id of the request message, as specified.
Exception Action	Discard DLQ	DLQ	Action when a message cannot be delivered. When set to DLQ, the message is sent to the dead-letter queue. When set to Discard, it is discarded.
Report Data	Report With Data With Full Data	Report	The amount of data included in a report message, where Report specifies no data, With Data specifies the first 100 bytes, and With Full Data specifies all data.

Table 15. Send attributes (continued)

Attribute	Values	Default	Comments
Report Type Exception	Selected Not selected	Not selected	When selected, Exception reports are required.
Report Type COA	Selected Not selected	Not selected	When selected, Confirm on Arrival reports are required.
Report Type COD	Selected Not selected	Not selected	When selected, Confirm on Delivery reports are required.
Report Type Expiry	Selected Not selected	Not selected	When selected, Expiry reports are required.
Segmentation	Selected Not selected	Not selected	When selected, Segmentation of the message is allowed.
Split File	Logical Physical	Physical	When set to Logical, the file is split into separate messages at record boundaries, as determined by the value of File Record Length. On Windows, HP-UX, AIX, and Sun Solaris, if the File Record Length is zero, this is the end of a line. On z/OS, this is a record boundary. When set to Physical, the file is split into separate messages on boundaries that are determined by AMI.
File Record Length	0-999999999	0	When Split File is set to Logical, a non-zero value specifies the boundary to use to split a file into individual messages. When Split File is set to Physical, this value is ignored.
Bind On Open	Yes No As Transport	As Transport	Bind On Open controls the binding of a service point to a particular instance of a WebSphere MQ cluster queue. When set to Yes, the service point is bound to the destination queue when the service is opened. When set to No, the service point is not bound to a specific destination, and successive sends using this service point may result in messages being sent to different instances of the destination queue. When set to As Transport, the value from the underlying queue definition is used.
Application Group	Name		Optional application group name used when the service represents a WebSphere MQ Integrator Version 1 broker. 2

Notes:

1 If Implicit Open is selected and Leave Open is not selected, MQPUT1 is used for send operations.

2 The name is a maximum of 256 characters, and can contain the following characters: `A-Z`, `a-z`, `0-9`, `'.'`, `'/'`, `'_'` and `'%'`.

Receive attributes

Table 16. Receive attributes

Attribute	Values	Default	Comments
Implicit Open	Selected Not selected	Selected	When selected, the queue is opened implicitly (must be selected for the C and COBOL high-level interfaces).
Leave Queue Open	Selected Not selected	Selected	When selected, a queue that was implicitly opened will be left open.
Delete On Close	Yes No Purge	No	When set to Yes, temporary dynamic queues, and permanent dynamic queues that contain no messages, are deleted when closed. When set to No, dynamic queues are not deleted when closed. When set to Purge, dynamic queues are deleted when closed, even if the queues contain messages.
Wait Interval	0-999999999 Unlimited	Unlimited	A period of time (in milliseconds) that the receive waits for a message to be available.
Wait Interval Read Only	Selected Not selected	Selected	When selected, an application cannot override the Wait Interval value in the policy object.
Convert	Selected Not selected	Selected	When selected, the message is code page converted by the message transport when received.
Wait For Whole Group	Selected Not selected	Selected	When selected, all messages in a group must be available before any message is returned by the receive. When not selected, AMRC_NO_MSG_AVAILABLE may be returned to the application before the complete group is received. In this case, any simulated group state information is destroyed and any remaining messages in a simulated group are orphaned.
Handle Poison Message	Selected Not selected	Selected	When selected, poison message handling is enabled. **1**
Accept Truncated Message	Selected Not selected	Not selected	When selected, truncated messages are accepted.
Open Shared	Selected Not selected	Selected	When selected, the queue is opened as a shared queue.
File Disposition	New Overwrite Append	New	Specifies whether an incoming file is created as a New file, Overwrites an existing file, or becomes an Append to an existing file.

Table 16. Receive attributes (continued)

Attribute	Values	Default	Comments
Note: **1** A poison message is one for which the count of the number of times it has been backed-out during a unit of work exceeds the maximum backout limit specified by the underlying WebSphere MQ transport queue object. If poison message handling is enabled during a receive request, the AMI handles it as follows: If a poison message is successfully requeued to the backout-requeue queue (specified by the underlying WebSphere MQ transport queue), the message is returned to the application with completion code MQCC_WARNING and reason code MQRC_BACKOUT_LIMIT_ERR. If a poison message requeue attempt (as described earlier) is unsuccessful, the message is returned to the application with completion code MQCC_WARNING and reason code MQRC_BACKOUT_REQUEUE_ERR. If a poison message is part of a message group (and not the only message in the group), no attempt is made to requeue the message. The message is returned to the application with completion code MQCC_WARNING and reason code MQRC_GROUP_BACKOUT_LIMIT_ERR. In all cases, a warning is returned and the message is returned to the application (even if it was successfully queued on the backout-requeue queue). Also, the message does not disappear from the original queue from where it is received, unless the application explicitly performs a commit.			

Subscribe attributes

Table 17. Subscribe attributes

Option	Values	Default	Comments
Subscribe Locally	Selected Not selected	Not selected	When selected, the subscriber is sent publications that were published with the Publish Locally option, at the local broker only.
New Publications Only	Selected Not selected	Not selected	When selected, the subscriber is not sent existing retained publications when it registers.
Publish On Request Only	Selected Not selected	Not selected	When selected, the subscriber is not sent retained publications, unless it requests them by using Request Update.
Inform If Retained	Selected Not selected	Selected	When selected, the broker informs the subscriber if a publication is retained.
Unsubscribe All	Selected Not selected	Not selected	When selected, all topics for this subscriber are to be deregistered.
Anonymous Registration	Selected Not selected	Not selected	When selected, the subscriber registers anonymously.
Use Correl Id As Id	Selected Not selected	Not selected	When selected, the Correl Id is used by the broker as part of the subscriber's identity.
Delivery Persistence	Persistent Non Persistent As Published As Transport	As Published	This controls the persistence of messages sent from the broker and applies only to WebSphere MQ Integrator Version 2.
Subscription Point	String		The character string for the subscription point to which the subscription is to be attached. If not specified, the default subscription point is assumed. This applies only to WebSphere MQ Integrator Version 2.

Publish attributes

Table 18. Publish attributes

Option	Values	Default	Comments
Retain	Selected Not selected	Not selected	When selected, the publication is retained by the broker.
Publish To Others Only	Selected Not selected	Not selected	When selected, the publication is not sent to the publisher if it has subscribed to the same topic (used for conference-type applications).
Suppress Registration	Selected Not selected	Selected	When selected, implicit registration of the publisher is suppressed. (This attribute is ignored for WebSphere MQ Integrator Version 2.)
Publish Locally	Selected Not selected	Not selected	When selected, the publication is only sent to subscribers that are local to the broker.
Accept Direct Requests	Selected Not selected	Not selected	When selected, the publisher should accept direct requests from subscribers.
Anonymous Registration	Selected Not selected	Not selected	When selected, the publisher registers anonymously.
Use Correl Id As Id	Selected Not selected	Not selected	When selected, the Correl Id is used by the broker as part of the publisher's identity.

Handler attributes

Table 19. Handler attributes

Attribute	Comments
Handler	The name that is assigned to this policy handler when it is created.
Invocation Parameters	A free-format text parameter string that is associated with the policy and passed to the policy handler library on each invocation to provide policy-specific information. The implementor of the policy handler defines the meaning and syntax of the string. The string is specific to the policy handler library, and the policy handler library parses and interprets this string. A typical use of this field is to pass policy-specific options to the policy handler library.

Policy handler definitions

This section describes the attributes for a policy handler library definition.

Policy handler attributes

Table 20. Policy Handler attributes

Attribute	Comments
Name	Mandatory policy handler name. Used to identify this policy handler on the list of handlers that are associated with a policy. **1**
Library	The name of the policy handler library file, excluding any directory information, prefix, and platform-specific extension, that the AMI should load for this policy handler. For AIX, HP-UX, and Solaris, the AMI prefixes `lib` to the specified name. The AMI appends the platform-dependent file extension to the specified name as follows: **.a** AIX (non-threaded) **_r.a** AIX (threaded) **_r** iSeries (threaded) **_r.sl** HP-UX **.so** Solaris **.dll** Windows and z/OS For iSeries non-threaded, there is no file extension.
Initialization Parameters	A free-format text parameter string that is passed to the policy handler library on initialization. The implementor of the policy handler defines the meaning and syntax of the string. The string is specific to the policy handler library, and the policy handler library parses and interprets this string.

Notes:

1 The name is a maximum of 256 characters, and can contain the following characters: `A-Z`, `a-z`, `0-9`, `'.'`, `'/'`, `'_'` and `'%'`.

Chapter 19. Lightweight Directory Access Protocol support

Information that describes the various users, applications, and other resources that are available on a computer network can be collected into a specialized database or repository called a directory.

Directories are usually accessed using a client/server computing model:
- Applications that need to access and update information in the directory issue requests to a directory server.
- The server manages the storage and retrieval of data in the directory.

The Lightweight Directory Access Protocol (LDAP) is an open industry standard that defines a protocol for the requests and responses that flow between directory clients and servers. AMI support for LDAP means that service, policy, and policy handler definitions can be stored and accessed across networks by using a directory as an alternative to distributing repository files.

It is possible to update local or remote LDAP directories with AMI information from a repository file. On Windows NT and Windows 2000, you can use the AMI Administration Tool to do this.

Environment variables are used to configure AMI applications to retrieve AMI information from the directory. A directory schema defines the objects and attributes that may be stored in and retrieved from a directory. The AMI includes schema definitions that provide object classes and attributes for AMI service (service point, distribution list, publisher, subscriber), policy, and policy handler objects.

AMI Version 1.2 supports LDAP on all AMI platforms. z/OS and iSeries operating systems include LDAP client software that is used by AMI. On the other platforms, the IBM SecureWay® Directory Version 3.2 Client (or later) must be installed. The schema definition files and information update facilities that are provided with the AMI are supported on the following LDAP directory servers:
- LDAP V3 IBM SecureWay Directory
- Microsoft Active Directory

These are described in the following sections.

Notes:
1. For z/OS, a suitable version of the SecureWay Directory Client is supplied with OS/390 Version 2 Release 9 or later.
2. On z/OS, you cannot access an LDAP repository directly from an AMI session. However, a direct input utility is provided to generate a repository cache directly from LDAP. See “Repository and local host caches” on page 472.

Getting started with LDAP

If your directory server does not have the AMI LDAP schema installed, install this first. See “Installation” on page 507. Note that you do not need to install the AMI on the machine that hosts the directory server.

To use the AMI with an LDAP directory:

LDAP support

1. Create your AMI repository xml file in the normal way using the AMI Administration Tool.
2. Update your directory server with the AMI repository information.

   You can do this directly from the AMI Administration Tool on Windows, or on any platform by using the xml repository file with the `amtldup` command line program. See "Updating LDAP from a repository" on page 509.

To access AMI repository information in a SecureWay or Active Directory when using your AMI application, you must set the AMT_REPOSITORY environment variable to an LDAP URL. See "Directory search" on page 512. Alternatively, in C++ or Java, you can use the **AmSessionFactory setRepository** method to specify the LDAP URL. See "setRepository" on page 201 (C++) or "setRepository" on page 407 (Java).

SecureWay Directory

SecureWay Directory is a directory service that is based on LDAP and DB2. SecureWay Directory at LDAP V3 level is available on AIX, OS/400 (Version 4 Release 5), OS/390 (Version 2 Release 9 or later), Solaris, and Windows NT platforms.

The SecureWay Directory supports the Simple Authentication and Security Layer (SASL) by using Kerberos or the CRAM-MD5 mechanism for authentication. The Secure Sockets Layer (SSL) or Transport Layer Security (TLS) can be used for transport security and authentication.

The AMI includes schema definition files for all the SecureWay Directory platforms.

| **Note:** No schema definition files are provided with the z/OS AMI product. The schema files will need to be extracted from the Windows AMI product. They can be found in the `amt\ldap` libraries after installation of the Program Files and DLLs.

Active Directory

The Active Directory is a directory service that is included with the Windows 2000 Server.

The Active Directory service provides an interface that supports LDAP V3. There is also a Component Object Model (COM) based interface, Active Directory Service Interfaces (ADSI). ADSI enables client applications to communicate with any directory services that are compliant with LDAP or Novell Directory Services (NDS). Active Directory is available only on Windows 2000 domain controllers.

The Active Directory provides SASL negotiation, using Kerberos as the default mechanism. SSL can be used for encryption.

The AMI includes a schema definition file to extend the Active Directory schema for AMI use.

Installation

For the AMI platforms, the SecureWay Directory Version 3.2 Client (or later) is available as a free download from the IBM Web site, on:

```
www.ibm.com/software/network/directory
```

For z/OS, a suitable version of IBM SecureWay Directory Client is supplied with OS/390 Version 2 Release 9 or later.

For each computer where you wish to use the AMI LDAP facilities, the SecureWay Client must be installed and available. This includes the computer from where you install the schema, and the computer from where you update the directory.

For the directory server platforms, there are schema extension files and scripts to install the schema. These files and scripts vary according to the directory service, not the operating system. Generally, each directory server that will be used to hold AMI information must have the AMI schema extensions installed once, usually by an administrator of the directory. For a Windows 2000 forest, the AMI schema extensions are installed once to extend the Active Directory schema, then they will be replicated throughout the forest.

If you use a version of the SecureWay Directory that is later than Version 3.2.1, the AMI schema is installed as part of the base schema.

Note that the target directory server itself must be configured with a suffix and initial data before AMI data can be stored in, or retrieved from, the directory.

SecureWay Directory

The AMI includes files that contain the AMI schema classes and attributes. If the AMI schema is not installed as part of the base schema (which depends on the version of the SecureWay Directory Server installed), you can use these files to install the AMI schema to the directory server.

If the directory schema is held under a suffix other than cn=schema (the default), you must modify the schema file to reference the appropriate suffix.

To install the AMI schema to a directory server on AIX, iSeries, Solaris, Windows, or z/OS run the **ldapmodify** program, which is provided with SecureWay Directory Client software.

Note: On OS/390, if you have an RDBM (DB2 backend) installation, see “RDBM installations” on page 508 instead of using the **ldapmodify** program. OS/390 Version 2 Release 10 and later supports both TDBM and RDBM. OS/390 Version 2 Release 9 supports RDBM only.

Using the ldapmodify program

1. Change directory to the AMI `ldap` directory (see “Directory structure” on page 445 (AIX), page 449 (HP-UX), page 458 (Solaris), or page 462 (Windows).
2. Issue the `ldapmodify` command:

   ```
   ldapmodify -h host -p port -D dn -w password -f filename
   ```

   where:

host	The TCP address of the computer on which the directory server is running (optional). The default value is localhost.
port	The port that the directory server uses (optional). The default value is 389.
dn	The distinguished name (DN) of a directory administrator. This name is used to bind to the directory.
password	The password for the distinguished name of the directory administrator.
filename	The file that contains the definitions AMI schema classes and attributes. For directory servers that run on AIX, iSeries, Solaris, or Windows, this is `amtsw.ldif`. For z/OS servers running TBDM, this is `amtsw390.ldif`.

If the directory server is configured to require high security, you can use further **ldapmodify** options. For further details about the **ldapmodify** command, refer to the SecureWay Directory documentation.

RDBM installations

This section describes how to install the AMI schema of the SecureWay LDAP server for RDBM installations on OS/390.

Two schema configuration files are created in the AMI samples directory at installation. These files are:

- `amtsw390.at`, which defines the AMI LDAP attributes
- `amtsw390.oc`, which defines the AMI LDAP object classes

To extend the directory, the administrator must:

1. Stop the directory server.
2. Transfer the RDBM schema definition files as text to the OS/390 Hierarchical File System, that is:
   - `amtsw390.at` to `/etc/ldap/amtsw390.at`
   - `amtsw390.oc` to `/etc/ldap/amtsw390.oc`
3. Edit the Secureway LDAP server configuration file `/etc/ldap/slapd.conf` to add the following two lines:

   ```
   include /etc/ldap/amtsw390.oc
   include /etc/ldap/amtsw390.at
   ```
4. Restart the Secureway LDAP server.

Active Directory

Installation includes a script, `admqami.vbs`, and the AMI schema classes and attributes in the file `admqami.ldf`. To extend the directory schema, you must log on to an appropriate Windows 2000 domain as a domain administrator, then run the supplied script.

Note: Active Directory schema changes are generally permanent and cannot be removed.

Changes to the Active Directory schema are replicated throughout the domain forest, so usually, you need to perform the schema update only once for each enterprise.

To extend the directory schema, use the following steps:

1. Ensure that the current Schema Master for the Active Directory is online.

   You can find and change the Schema Master by using the Active Directory Schema MMC snap-in. This is provided with Windows 2000 Server and Windows 2000 Advanced Server. For details, refer to the Microsoft documentation about the Active Directory. The snap-in also indicates whether the schema master is available.
2. On the domain controller where you wish to initiate the schema update (this does not need to be the Schema Master), ensure that the following registry flag is set to 1:

   ```
     HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\
   Parameters\Schema Update Allowed
   ```

   To do this, either use the Active Directory Schema MMC snap-in, or use the **regedit** command.

   This setting means that the schema can be modified from the selected domain controller, or from a workstation in the domain.
3. Log on as a domain administrator to a computer on which AMI is installed. This computer must be a Windows 2000 domain controller or Windows 2000 Professional computer.
4. At a command prompt, change directory to the AMI ldap directory (see “Directory structure (Windows)” on page 462).
5. Run the following script file: `amtad.vbs`

   This script determines the current Schema Master and runs the Microsoft LDIFDE utility to add definitions from the admqami.ldf file to the Active Directory schema. When prompted, select **Yes** to confirm that you wish to update the schema.

   If the update fails, the script displays the error code that the Microsoft LDIFDE utility returns.

Uninstallation

If you uninstall WebSphere MQ, there are no special considerations or actions required for LDAP. Entries that are created in target directories are not removed. If required, you can delete schema extensions from the SecureWay Directory manually. However, for the Active Directory, you can only disable the schema extensions.

Updating LDAP from a repository

It is possible to update local or remote LDAP directories with AMI information from a repository file. On Windows NT and Windows 2000, you can use the AMI Administration Tool to do this. From the AMI Administration Tool, you can update a remote LDAP directory on any supported platform, for example, z/OS or Solaris.

On AIX, HP-UX, Solaris, and Windows, you can update an LDAP directory using the command line.

Using the AMI Administration Tool

To use the AMI Administration Tool to save the current repository file to a LDAP directory:

1. Start the AMI Administration Tool (see "The AMI Administration Tool" on page 475).
2. Select **File→Update LDAP Directory**.

   If the information is not saved to a file, it is automatically saved, or you are prompted to save it to a new file.
3. When the Update LDAP Directory window is displayed, enter the following information:

LDAP Server Name	The TCP address of the LDAP server host (default, localhost).
Port	The port to connect to the LDAP server (default 389).
Base Dn	The distinguished name under which the AMI information is placed in the Directory Information Tree. The default is the container cn=AMIInfo under the default directory naming context.
Authentication Type	The method to use to log in to the directory. Select from: • None • Simple (the default) • CRAM MD5 • Kerberos • External
Bind dn	The user Id. This user must have permission to create entries in the target directory. If the Authentication Type is set to None, Kerberos, or External, you can leave this field blank.
Password	Password for the user Id. If the Authentication Type is set to None, Kerberos, or External, you can leave this field blank.
Use SSL	Select this option if Secure Sockets Layer (SSL) is used.
Key File Name	The file that contains the encryption keys to use with SSL. Optionally, use the Browse button to select the file. If you do not select Use SSL, you can leave this field blank.
Key File Password	The password for the key file. If you do not select Use SSL, you can leave this field blank.
Private Key Name	The private key name to use in the key file.

	If you do not select Use SSL, you can leave this field blank.
LDAP Version	The version of the LDAP server (2 or 3).

4. When your entries are complete, click **OK**. The AMI LDAP update program runs and updates the specified directory with information from the repository file.

   When the operation completes successfully, a success message is displayed. Otherwise, a message that includes the error code is displayed. For details about any error messages, see "LDAP error codes" on page 556, or the AMI Administration Tool online help.

Using the command line

The AMI LDAP Directory Update Program, amtldup, is provided so that you can update a directory from a repository xml file using the command line, and on platforms other than Windows.

To use the command, change to the directory that contains the amtldup command, then enter the command with the required options. For the location of the amtldup executable file on the relevant platform, see "Directory structure" on page 445 (AIX), page 449 (HP-UX), page 458 (Solaris), or page 462 (Windows).

The syntax of the amtldup command is:

```
amtldup [options] -f file
```

where the options are:

-h *host*	The TCP address of the LDAP server host (default, localhost).
-p *port*	The port to connect to the LDAP server (default 389).
-D *dn*	The user Id in Distinguished Name form. This user must have permission to create entries in the Directory Information Tree.
-w *password*	The password for the user Id.
-Z	Use a secure LDAP connection (SSL).
-K *keyfile*	The name of the file that contains the encryption keys for use with SSL. This option is required if `-Z` is specified.
-P *key_pw*	The password of the key file. This option is required if `-Z` is specified.
-N *key_name*	The private key name to use in the key file. This option is required if `-Z` is specified.
-m *mechanism*	The Simple Authentication and Security Layer (SASL) authentication mechanism to use to bind to the directory. If specified, the mechanism must be one of: • `CRAM-MD5` for CRAM MD5 authentication • `GSSAPI` for Kerberos authentication • `EXTERNAL` for any other user-provided SASL mechanism
-V *version*	The version of the LDAP server (2 or 3; default is 3).
-b *base_dn*	The base Distinguished Name under which new entries are created. The default is the container cn=AMIInfo under the default directory naming context.

-v Use verbose mode.

-? Display syntax and options.

and `file` is the name of the repository xml file.

To delete AMI entries under the *base_dn*, use:

```
amtldup [options] -X
```

Directory information tree

Entries in a directory are organized in a treelike structure called the Directory Information Tree (DIT). Within the DIT, the Distinguished Name (DN) of each entry determines its position in the tree. To store AMI information in the directory, you must select a base DN for the data. If the base DN does not already exist, it is created, then all the AMI data is placed under the base. Similarly, when the AMI searches for services and policies, you must specify the base DN from which to start the search. Typically, the contents of each XML repository file are stored under a distinct base DN (that is, in a separate container).

By default, the base DN is a container with the name cn=AMIInfo under the directory suffix.

The directory administrator must manage authorization to use the directory objects by using the usual administration utilities. The AMI does not provide tools to manage access control lists for service and policy information. When the entries are created, they inherit the access control list from the parent object. For further information, see “Security” on page 513.

Directory search

When an AMI application specifies a service or policy name, the AMI searches either the local repository file, or an LDAP directory. This depends on how the environment variables that AMI uses are set.

To search an LDAP directory for repository information, the AMT_REPOSITORY environment variable must be set to a reference to the repository information in LDAP URL format. This format is specified in RFC 2255 (for details, refer to the IETF Request for Comments Web page at `www.ietf.org/rfc.html`). For example:

```
  AMT_REPOSITORY=ldap://ldap.hursley.ibm.com:389/cn=Rep1,cn=AMIInfo,
dc=hursley,o=ibm,c=uk
```

The LDAP URL can contain the bind DN, that is, the user Id with which to bind (this is needed for simple and CRAM-MD5 authentication). For example:

```
  AMT_REPOSITORY=ldap://ldap.hursley.ibm.com:389/cn=Rep1,cn=AMIInfo,
dc=hursley,o=ibm,c=uk??sub??bindname=cn=Manager,o=MQSeries,o=ibm,c=uk
```

Alternatively, you can specify the bind DN separately using the AMT_LDAP_BINDDN environment variable, for example:

```
AMT_LDAP_BINDDN=cn=Manager,o=MQSeries,o=ibm,c=uk
```

If you specify the bind DN using both the LDAP URL and the AMT_LDAP_BINDDN environment variable, the environment variable takes precedence.

For C++ and Java applications, the LDAP URL can be specified by using the **AmSessionFactory setRepository** method. See "setRepository" on page 201 (C++) or "setRepository" on page 407 (Java).

To set security parameters, use the following environment variables:

AMT_LDAP_AUTHENTICATION
Set to one of none, simple, CRAM-MD5, or Kerberos according to the required security mechanism. If there is no bind DN set, the default is none, which implies anonymous access. Otherwise, the default is simple authentication.

AMT_LDAP_PASSWORD
Set to the password for the bind DN, if a password is required for the security mechanism that AMT_LDAP_AUTHENTICATION sets.

AMT_SSL
If this environment variable is set, Secure Sockets Layer (SSL) is used. Set it to the key file name, password, and key name (filename, password, name), if required. For example,

```
AMT_SSL=keyfile,secret,cn=Manager,o=MQSeries,o=ibm,c=uk
```

Security

AMI administrators must consider the security of the bind DN. This is the user Id with which the AMI binds to the directory to extend, and search for, service and policy information. Authorization can range between full anonymous access, and the allocation of individual identities and access authorizations to each user.

For any chosen security strategy, the appropriate identities and passwords must be defined in the directory, and set by using the AMI environment variables on the client computers. You can set authorization to access the AMI information for individual users, or for groups, by using the normal directory utilities. You can use the Access Control List and inheritance mechanism of the directory to simplify authorization.

For SSL authentication or encryption, the IBM Global Security Kit (GSKit) V4 or later must be installed on each client machine that uses the AMI or the AMI Administration Tool). This is available as a free download from the IBM Web site, on:

```
www.ibm.com/software/network/directory
```

For Kerberos, the Kerberos client, IBM Network Authentication Service, must be installed. Note that Kerberos is not supported on Solaris and HP-UX.

LDAP support

Chapter 20. Problem determination

This chapter shows you how to use the trace facility in the Application Messaging Interface, and gives some information about finding the causes of problems. See:

- "Using trace (iSeries, UNIX, and Windows)"
- "Using trace (z/OS)" on page 529
- "When your AMI program fails" on page 532

Using trace (iSeries, UNIX, and Windows)

The Application Messaging Interface includes a trace facility to help identify what is happening when you have a problem. It shows the paths taken when you run your AMI program. Unless you have a problem, you are recommended to run with tracing set off to avoid any unnecessary overheads on your system resources.

There are three environment variables that you set to control trace:

AMT_TRACE

AMT_TRACE_PATH

AMT_TRACE_LEVEL

For iSeries, you set these environment variables using the following commands:

ADDENVVAR - Adds an environment variable

CHGENVVAR - Changes an environment variable

WRKENVVAR - Displays an environment variable

RMVENVVAR - Deletes an environment variable

To set global environment variables, specify LEVEL(*SYS) in the ADDENVAR command, for example:

```
ADDENVVAR ENVVAR(variable) VALUE(value) LEVEL(*SYS)
```

Alternatively, you can create a CL program that contains commands to set the environment variables. At startup, you can run this program by specifying the name of the CL program with the SYSVAL QSTRUPPGM command, for example:

```
CHGSYSVAL SYSVAL(QSTRUPPGM) VALUE('program')
```

For UNIX or Windows, you set these variables in one of two ways.

1. From a command prompt. The settings are locally effective, so you must then start your AMI program from this prompt.

   If you use the export command with the OS/400 Qshell interpreter, you must specify the `-s` option to set the environment in the current process.
2. By putting the information into your system startup file. These settings are globally effective. To do this:
   - On Windows, select **Start->Settings->Control Pane**l, select **System**, select the **Environment** tab, then add or set the environment variables.
   - On UNIX systems, edit your `.profile` file.

When deciding where you want the trace files written, ensure that the user has sufficient authority to write to, not just read from, the disk.

Using trace (iSeries, UNIX, and Windows)

If you have tracing switched on, it will slow down the running of your AMI program, but it will not affect the performance of your WebSphere MQ environment. When you no longer need a trace file, it is your responsibility to delete it. You must stop your AMI program running to change the status of the AMT_TRACE variable. The AMI trace environment variable is different than the trace environment variable used within the WebSphere MQ range of products. Within the AMI, the trace environment variable turns tracing on. If you set the variable to a string of characters (any string of characters) tracing will remain switched on. It is not until you set the variable to NULL that tracing is turned off.

Trace filename and directory

The trace file name takes the form AMTnnnnn.trc, where nnnnn is the ID of the AMI process running at the time.

Commands on iSeries

WRKENVVAR
Displays the settings of all environment variables.

ADDENVVAR ENVVAR(AMT_TRACE_PATH) VALUE('/directory')
Sets the trace directory where the trace file will be written.

RMVENVVAR ENVVAR(AMT_TRACE_PATH)
Removes the AMT_TRACE_PATH environment variable; the trace file is written to the current working directory (when the AMI program was started).

ADDENVVAR ENVVAR(AMT_TRACE_LEVEL) VALUE(n)
Sets the trace level, where n is an integer from 0 through 9. 0 represents minimal tracing, and 9 represents a fully detailed trace.

You can also suffix the value with a + (plus) or - (minus) sign. When the plus sign is suffixed, the trace includes all control block dump information and all informational messages. When the minus sign is suffixed, the trace includes only the entry and exit points in the trace, with no control block information or text output to the trace file.

RMVENVVAR ENVVAR(AMT_TRACE_LEVEL)
Removes the AMT_TRACE_LEVEL environment variable. The trace level is set to its default value of 2.

ADDENVVAR ENVVAR(AMT_TRACE) VALUE(xxxxxxxx)
Sets tracing ON by putting one or more characters for the VALUE parameter. For example:

```
ADDENVVAR  ENVVAR(AMT_TRACE)  VALUE(yes)
ADDENVVAR  ENVVAR(AMT_TRACE)  VALUE(no)
```

In both of these examples, tracing will be set ON.

RMVENVVAR ENVVAR(AMT_TRACE)
Sets tracing off.

Commands on UNIX

export AMT_TRACE_PATH=/directory
Sets the trace directory where the trace file will be written.

unset AMT_TRACE_PATH
Removes the AMT_TRACE_PATH environment variable; the trace file is written to the current working directory (when the AMI program was started).

echo $AMT_TRACE_PATH
Displays the current setting of the trace directory path.

export AMT_TRACE_LEVEL=n
Sets the trace level, where `n` is an integer from 0 through 9. 0 represents minimal tracing, and 9 represents a fully detailed trace.

You can also suffix the value with a + (plus) or - (minus) sign. When the plus sign is suffixed, the trace includes all control block dump information and all informational messages. When the minus sign is suffixed, the trace includes only the entry and exit points in the trace, with no control block information or text output to the trace file.

unset AMT_TRACE_LEVEL
Removes the AMT_TRACE_LEVEL environment variable. The trace level is set to its default value of 2.

echo $AMT_TRACE_LEVEL
Displays the current setting of the trace level.

export AMT_TRACE=xxxxxxxx
Sets tracing ON by putting one or more characters after the ‘=’ sign. For example:

```
export  AMT_TRACE=yes
export  AMT_TRACE=no
```

In both of these examples, tracing will be set ON.

unset AMT_TRACE
Sets tracing off.

echo $AMT_TRACE
Displays the contents of the environment variable.

Commands on Windows

SET AMT_TRACE_PATH=drive:\directory
Sets the trace directory where the trace file will be written.

SET AMT_TRACE_PATH=
Removes the AMT_TRACE_PATH environment variable; the trace file is written to the current working directory (when the AMI program was started).

SET AMT_TRACE_PATH
Displays the current setting of the trace directory.

SET AMT_TRACE_LEVEL=n
Sets the trace level, where `n` is an integer from 0 through 9. 0 represents minimal tracing, and 9 represents a fully detailed trace.

You can also suffix the value with a + (plus) or - (minus) sign. When the plus sign is suffixed, the trace includes all control block dump information and all informational messages. When the minus sign is suffixed, the trace includes only the entry and exit points in the trace, with no control block information or text output to the trace file.

SET AMT_TRACE_LEVEL=
Removes the AMT_TRACE_LEVEL environment variable. The trace level is set to its default value of 2.

SET AMT_TRACE_LEVEL
Displays the current setting of the trace level.

SET AMT_TRACE=xxxxxxxx
Sets tracing ON by putting one or more characters after the '=' sign. For example:

```
SET  AMT_TRACE=yes
SET  AMT_TRACE=no
```

In both of these examples, tracing will be set ON.

SET AMT_TRACE=
Sets tracing OFF.

SET AMT_TRACE
Displays the contents of the environment variable.

C++ and Java

For these language bindings, there is more control over the production of trace. In each case, the AmSessionFactory has two methods that control trace:
1. setTraceLocation(location);
2. setTraceLevel(level);

The behavior of these methods matches exactly the behavior of the environment variables:
1. AMT_TRACE_PATH
2. AMT_TRACE_LEVEL

Once an AmSession has been created using an AmSessionFactory, the trace level and location are set for the complete life of that AmSession.

If set, the values of the properties in the AmSessionFactory take precedence over any AMT trace environment variables.

Example trace

The following example trace shows ‘typical’ trace output.

```
Trace for program E:\users\winn\build\bin\amtsosnd.exe <<< AMT trace >>>
started at Wed May 30 09:07:10 2001

@(!) <<< *** Code Level is 1.3.0 *** >>>
   !(00330) BuildDate May 29 2001
   !(00330) Trace Level is 9
(00330)@09:07:10.513
   -->xmq_xxxInitialize
   ---->ObtainSystemCp
   !(00330) Code page is 437
   <----ObtainSystemCp (rc = 0)
   <--xmq_xxxInitialize (rc = 0)
   -->amSessCreateX
   ---->amCheckAllBlanks()
   <----amCheckAllBlanks() (rc = 0)
   ---->amCheckValidName()
   <----amCheckValidName() (rc = 1)
   !(00330) Session name is: AMT.SAMPLE.SESSION
   !(00330) Allocating Object lock
   !(00330) Allocated object lock 008A2FC0
   !(00330) amLOCK_OBJECT_INIT(008A2FC0)
   ---->amIdxTableAddEntry
   ------>amIdxTableCreate
   !(00330) allocating 1076
   !(00330) amLOCK_GLOBAL() 0
   !(00330) amLOCK_OBJECT_INIT(009BFD68)
   !(00330) amUNLOCK_GLOBAL() 0
   <------amIdxTableCreate (rc = AM_ERR_OK)
   ------>amIdxTableLock
   !(00330) amLOCK_OBJECT(009BFD68) 0
   <------amIdxTableLock (rc = AM_ERR_OK)
   ------>amIdxTableUnlock
   !(00330) amLOCK_OBJECT(009BFD68) 0
   <------amIdxTableUnlock (rc = AM_ERR_OK)
   <----amIdxTableAddEntry (rc = AM_ERR_OK)
   ---->amSesClearErrorCodes
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(009BFD68) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(009BFD68) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 008A2780
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(009BFD68) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(009BFD68) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   ------>amErrTranslate
   <------amErrTranslate (rc = 0)
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(009BFD68) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(009BFD68) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   <----amSesClearErrorCodes (rc = 0)
   ---->amHashTableCreate()
   ------>amHashTableInit()
(00330)@09:07:10.604
   <------amHashTableInit() (rc = AM_ERR_OK)
   <----amHashTableCreate() (rc = AM_ERR_OK)
   ---->amHashTableCreate()
   ------>amHashTableInit()
   <------amHashTableInit() (rc = AM_ERR_OK)
   <----amHashTableCreate() (rc = AM_ERR_OK)
   ---->amHashTableCreate()
   ------>amHashTableInit()
   <------amHashTableInit() (rc = AM_ERR_OK)
   <----amHashTableCreate() (rc = AM_ERR_OK)
   ---->amHashTableCreate()
   ------>amHashTableInit()
   <------amHashTableInit() (rc = AM_ERR_OK)
   <----amHashTableCreate() (rc = AM_ERR_OK)
   ---->amHashTableCreate()
   ------>amHashTableInit()
   <------amHashTableInit() (rc = AM_ERR_OK)
   <----amHashTableCreate() (rc = AM_ERR_OK)
   ---->amHashTableCreate()
   ------>amHashTableInit()
   <------amHashTableInit() (rc = AM_ERR_OK)
   <----amHashTableCreate() (rc = AM_ERR_OK)
   ---->amHashTableCreate()
   ------>amHashTableInit()
   <------amHashTableInit() (rc = AM_ERR_OK)
   <----amHashTableCreate() (rc = AM_ERR_OK)
   ---->amHashTableCreate()
   ------>amHashTableInit()
   <------amHashTableInit() (rc = AM_ERR_OK)
   <----amHashTableCreate() (rc = AM_ERR_OK)
   ---->amMaSrvCreate
   !(00330) amLOCK_GLOBAL() 0
   ------>amIdxTableCreate
   !(00330) allocating 1076
   !(00330) amLOCK_GLOBAL() 1
   !(00330) amLOCK_OBJECT_INIT(0089B600)
   !(00330) amUNLOCK_GLOBAL() 1
   <------amIdxTableCreate (rc = AM_ERR_OK)
   !(00330) amUNLOCK_GLOBAL() 0
   !(00330) Service object created 0x899CB8
   ------>amIdxTableAddEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableAddEntry (rc = AM_ERR_OK)
   ------>amMaSrvClearErrorCodes
   -------->amIdxTableGetEntry
(00330)@09:07:10.754
   ---------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableLock (rc = AM_ERR_OK)
   ---------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableUnlock (rc = AM_ERR_OK)
   <--------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 00899CB8
   <------amMaSrvClearErrorCodes (rc = AM_ERR_OK)
   <----amMaSrvCreate (rc = AM_ERR_OK)
   ---->amMaSrvSetSessionHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 00899CB8
   <----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
   ---->amHashTableAddHandle()
   ------>amHashTableAddElement()
   !(00330) Element [SYSTEM.DEFAULT.SENDER] slot [61]
   <------amHashTableAddElement() (rc = AM_ERR_OK)
   <----amHashTableAddHandle() (rc = AM_ERR_OK)
   ---->amMaSrvCreate
   !(00330) Service object created 0x89BA48
   ------>amIdxTableAddEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableAddEntry (rc = AM_ERR_OK)
   ------>amMaSrvClearErrorCodes
   -------->amIdxTableGetEntry
   ---------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableLock (rc = AM_ERR_OK)
   ---------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableUnlock (rc = AM_ERR_OK)
   <--------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 0089BA48
   <------amMaSrvClearErrorCodes (rc = AM_ERR_OK)
   <----amMaSrvCreate (rc = AM_ERR_OK)
   ---->amMaSrvSetSessionHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
(00330)@09:07:10.814
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 0089BA48
   <----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
   ---->amHashTableAddHandle()
   ------>amHashTableAddElement()
   !(00330) Element [SYSTEM.DEFAULT.RESPONSE.SENDER] slot [69]
   <------amHashTableAddElement() (rc = AM_ERR_OK)
   <----amHashTableAddHandle() (rc = AM_ERR_OK)
   ---->amMaSrvCreate
   !(00330) Service object created 0x89D390
   ------>amIdxTableAddEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableAddEntry (rc = AM_ERR_OK)
   ------>amMaSrvClearErrorCodes
   -------->amIdxTableGetEntry
   ---------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableLock (rc = AM_ERR_OK)
   ---------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableUnlock (rc = AM_ERR_OK)
   <--------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 0089D390
   <------amMaSrvClearErrorCodes (rc = AM_ERR_OK)
   <----amMaSrvCreate (rc = AM_ERR_OK)
   ---->amMaSrvSetSessionHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 0089D390
   <----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
   ---->amHashTableAddHandle()
   ------>amHashTableAddElement()
   !(00330) Element [SYSTEM.DEFAULT.RECEIVER] slot [16]
   <------amHashTableAddElement() (rc = AM_ERR_OK)
   <----amHashTableAddHandle() (rc = AM_ERR_OK)
   ---->amMaSrvCreate
   !(00330) Service object created 0x9C01B0
   ------>amIdxTableAddEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableAddEntry (rc = AM_ERR_OK)
(00330)@09:07:10.824
   ------>amMaSrvClearErrorCodes
   -------->amIdxTableGetEntry
   ---------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableLock (rc = AM_ERR_OK)
   ---------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableUnlock (rc = AM_ERR_OK)
   <--------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C01B0
   <------amMaSrvClearErrorCodes (rc = AM_ERR_OK)
   <----amMaSrvCreate (rc = AM_ERR_OK)
   ---->amMaSrvSetSessionHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C01B0
   <----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
   ---->amHashTableAddHandle()
   ------>amHashTableAddElement()
   !(00330) Element [SYSTEM.DEFAULT.PUBLISHER] slot [34]
   <------amHashTableAddElement() (rc = AM_ERR_OK)
   <----amHashTableAddHandle() (rc = AM_ERR_OK)
   ---->amMaSrvCreate
   !(00330) Service object created 0x9C1AF8
   ------>amIdxTableAddEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableAddEntry (rc = AM_ERR_OK)
   ------>amMaSrvClearErrorCodes
   -------->amIdxTableGetEntry
   ---------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableLock (rc = AM_ERR_OK)
   ---------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableUnlock (rc = AM_ERR_OK)
   <--------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C1AF8
   <------amMaSrvClearErrorCodes (rc = AM_ERR_OK)
   <----amMaSrvCreate (rc = AM_ERR_OK)
   ---->amMaSrvSetSessionHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
(00330)@09:07:10.824
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C1AF8
   <----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
   ---->amHashTableAddHandle()
   ------>amHashTableAddElement()
   !(00330) Element [SYSTEM.DEFAULT.SUBSCRIBER] slot [18]
   <------amHashTableAddElement() (rc = AM_ERR_OK)
   <----amHashTableAddHandle() (rc = AM_ERR_OK)
   ---->amMaSrvCreate
   !(00330) Service object created 0x9C3440
   ------>amIdxTableAddEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableAddEntry (rc = AM_ERR_OK)
   ------>amMaSrvClearErrorCodes
   -------->amIdxTableGetEntry
   ---------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableLock (rc = AM_ERR_OK)
   ---------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <----------amIdxTableUnlock (rc = AM_ERR_OK)
   <--------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C3440
   <------amMaSrvClearErrorCodes (rc = AM_ERR_OK)
   <----amMaSrvCreate (rc = AM_ERR_OK)
   ---->amMaSrvSetSessionHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C3440
   <----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
   ---->amMaSrvSetSubReceiverHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C1AF8
   <----amMaSrvSetSubReceiverHandle (rc = AM_ERR_OK)
   ---->amMaMsgCreate
   !(00330) amLOCK_GLOBAL() 0
(00330)@09:07:11.124
   ------>amIdxTableCreate
   !(00330) allocating 1076
   !(00330) amLOCK_GLOBAL() 1
   !(00330) amLOCK_OBJECT_INIT(009C6740)
   !(00330) amUNLOCK_GLOBAL() 1
   <------amIdxTableCreate (rc = AM_ERR_OK)
   !(00330) amUNLOCK_GLOBAL() 0
   !(00330) message object created -[10243464]
   ------>amIdxTableAddEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableAddEntry (rc = AM_ERR_OK)
   ------>amMaMsgClearErrorCodes
   -------->amIdxTableGetEntry
   ---------->amIdxTableLock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <----------amIdxTableLock (rc = AM_ERR_OK)
   ---------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <----------amIdxTableUnlock (rc = AM_ERR_OK)
   <--------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C4D88
   <------amMaMsgClearErrorCodes (rc = AM_ERR_OK)
   <----amMaMsgCreate (rc = AM_ERR_OK)
   ---->amMaMsgSetSessionHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C4D88
   <----amMaMsgSetSessionHandle (rc = AM_ERR_OK)
   ---->amHashTableAddHandle()
   ------>amHashTableAddElement()
   !(00330) Element [SYSTEM.DEFAULT.SEND.MESSAGE] slot [83]
   <------amHashTableAddElement() (rc = AM_ERR_OK)
   <----amHashTableAddHandle() (rc = AM_ERR_OK)
   ---->amMaMsgCreate
   !(00330) message object created -[10259368]
   ------>amIdxTableAddEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableAddEntry (rc = AM_ERR_OK)
   ------>amMaMsgClearErrorCodes
   -------->amIdxTableGetEntry
   ---------->amIdxTableLock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <----------amIdxTableLock (rc = AM_ERR_OK)
   ---------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(009C6740) 0
(00330)@09:07:11.335
   <----------amIdxTableUnlock (rc = AM_ERR_OK)
   <--------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C8BA8
   <------amMaMsgClearErrorCodes (rc = AM_ERR_OK)
   <----amMaMsgCreate (rc = AM_ERR_OK)
   ---->amMaMsgSetSessionHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(009C6740) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009C8BA8
   <----amMaMsgSetSessionHandle (rc = AM_ERR_OK)
   ---->amHashTableAddHandle()
   ------>amHashTableAddElement()
   !(00330) Element [SYSTEM.DEFAULT.RECEIVE.MESSAGE] slot [17]
   <------amHashTableAddElement() (rc = AM_ERR_OK)
   <----amHashTableAddHandle() (rc = AM_ERR_OK)
   ---->amMaPolCreate
   !(00330) amLOCK_GLOBAL() 0
   ------>amIdxTableCreate
   !(00330) allocating 1076
   !(00330) amLOCK_GLOBAL() 1
   !(00330) amLOCK_OBJECT_INIT(0089F8F0)
   !(00330) amUNLOCK_GLOBAL() 1
   <------amIdxTableCreate (rc = AM_ERR_OK)
   !(00330) amUNLOCK_GLOBAL() 0
   !(00330) policy object created.
   !(00330) policy object initialized.
   ------>amIdxTableAddEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableAddEntry (rc = AM_ERR_OK)
   ------>amMaPolClearErrorCodes
   -------->amIdxTableGetEntry
   ---------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <----------amIdxTableLock (rc = AM_ERR_OK)
   ---------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <----------amIdxTableUnlock (rc = AM_ERR_OK)
   <--------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009CB968
   <------amMaPolClearErrorCodes (rc = AM_ERR_OK)
   <----amMaPolCreate (rc = AM_ERR_OK)
   ---->amMaPolSetSessionHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009CB968
(00330)@09:07:11.435
   <----amMaPolSetSessionHandle (rc = AM_ERR_OK)
   ---->amHashTableAddHandle()
   ------>amHashTableAddElement()
   !(00330) Element [SYSTEM.DEFAULT.POLICY] slot [29]
   <------amHashTableAddElement() (rc = AM_ERR_OK)
   <----amHashTableAddHandle() (rc = AM_ERR_OK)
   ---->amMaPolCreate
   !(00330) policy object created.
   !(00330) Setting syncpoint on in policy
   !(00330) policy object initialized.
   ------>amIdxTableAddEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableAddEntry (rc = AM_ERR_OK)
   ------>amMaPolClearErrorCodes
   -------->amIdxTableGetEntry
   ---------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <----------amIdxTableLock (rc = AM_ERR_OK)
   ---------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <----------amIdxTableUnlock (rc = AM_ERR_OK)
   <--------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009CC140
   <------amMaPolClearErrorCodes (rc = AM_ERR_OK)
   <----amMaPolCreate (rc = AM_ERR_OK)
   ---->amMaPolSetSessionHandle
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089F8F0) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 009CC140
   <----amMaPolSetSessionHandle (rc = AM_ERR_OK)
   ---->amHashTableAddHandle()
   ------>amHashTableAddElement()
   !(00330) Element [SYSTEM.DEFAULT.SYNCPOINT.POLICY] slot [80]
   <------amHashTableAddElement() (rc = AM_ERR_OK)
   <----amHashTableAddHandle() (rc = AM_ERR_OK)
   ---->amMaSrvSetStringProp
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableLock (rc = AM_ERR_OK)
   -------->amIdxTableUnlock
   !(00330) amLOCK_OBJECT(0089B600) 0
   <--------amIdxTableUnlock (rc = AM_ERR_OK)
   <------amIdxTableGetEntry (rc = AM_ERR_OK)
   !(00330) Object pointer 00899CB8
   !(00330) [AMSRV_STR_Q_NAME] set to [SYSTEM.DEFAULT.SENDER]
   <----amMaSrvSetStringProp (rc = AM_ERR_OK)
(00330)@09:07:11.445
   ---->amMaSrvSetStringProp
   ------>amIdxTableGetEntry
   -------->amIdxTableLock
```

```
!(00330) amLOCK_OBJECT(0089B600) 0
<--------amIdxTableLock (rc = AM_ERR_OK)
-------->amIdxTableUnlock
!(00330) amLOCK_OBJECT(0089B600) 0
<--------amIdxTableUnlock (rc = AM_ERR_OK)
<------amIdxTableGetEntry (rc = AM_ERR_OK)
!(00330) Object pointer 0089D390
!(00330) [AMSRV_STR_Q_NAME] set to [SYSTEM.DEFAULT.RECEIVER]
<----amMaSrvSetStringProp (rc = AM_ERR_OK)
---->amMaSrvSetStringProp
------>amIdxTableGetEntry
-------->amIdxTableLock
!(00330) amLOCK_OBJECT(0089B600) 0
<--------amIdxTableLock (rc = AM_ERR_OK)
-------->amIdxTableUnlock
!(00330) amLOCK_OBJECT(0089B600) 0
<--------amIdxTableUnlock (rc = AM_ERR_OK)
<------amIdxTableGetEntry (rc = AM_ERR_OK)
!(00330) Object pointer 009C01B0
!(00330) [AMSRV_STR_Q_NAME] set to [SYSTEM.DEFAULT.PUBLISHER]
<----amMaSrvSetStringProp (rc = AM_ERR_OK)
---->amMaSrvSetStringProp
------>amIdxTableGetEntry
-------->amIdxTableLock
!(00330) amLOCK_OBJECT(0089B600) 0
<--------amIdxTableLock (rc = AM_ERR_OK)
-------->amIdxTableUnlock
!(00330) amLOCK_OBJECT(0089B600) 0
<--------amIdxTableUnlock (rc = AM_ERR_OK)
<------amIdxTableGetEntry (rc = AM_ERR_OK)
!(00330) Object pointer 009C1AF8
!(00330) [AMSRV_STR_Q_NAME] set to [SYSTEM.DEFAULT.SUBSCRIBER]
<----amMaSrvSetStringProp (rc = AM_ERR_OK)
---->amActivateFiles
!(00330) No DATAPATH specified from API
------>amGetDataPath()
!(00330) Using environment[E:\MQSeries\amt\]
<------amGetDataPath() (rc = 1)
!(00330) No POL HANDLER PATH specified from API
------>amGetPolHandlerPath()
-------->amGetInstalledPath()
<--------amGetInstalledPath() (rc = 1)
!(00330) Using installPath[E:\MQSeries\amt\handlers\]
<------amGetPolHandlerPath() (rc = 1)
!(00330) Policy Handler Path E:\MQSeries\amt\handlers\ from Installed Path Used
!(00330) No repository FILE specified from API
------>amGetRepositoryName()
!(00330) Using default[amt.xml]
<------amGetRepositoryName() (rc = 1)
!(00330) Repository 0x8A3670
!(00330) Repository ACTIVE
!(00330) No local host FILE specified from API
------>amGetLocalHostFileName()
!(00330) Using default[amthost.xml]
<------amGetLocalHostFileName() (rc = 1)
!(00330) Local Host 0x8A3620
!(00330) Local Host File ACTIVE
<----amActivateFiles (rc = 1)
!(00330) Using repository.
---->amRpsCreate
!(00330) amLOCK_GLOBAL() 0
!(00330) Creating Repository Cache
------>CRpsTree::CRpsTree
<------CRpsTree::CRpsTree (rc = 0)
!(00330) amUNLOCK_GLOBAL() 0
<----amRpsCreate (rc = AM_ERR_OK)
---->amRpsGetIsOpen
```

```
   !(00330) Object handle[9056960]
(00330)@09:07:11.845
   <----amRpsGetIsOpen (rc = AM_ERR_OK)
   ---->amRpsOpen
   !(00330) amLOCK_GLOBAL() 0
   !(00330) Object handle[9056960]
   ------>CRpsTree::refresh
   -------->CRpsTree::clearContent
   <--------CRpsTree::clearContent (rc = 0)
   !(00330) repository file name: E:\MQSeries\amt\amt.xml
   -------->CRpsTree::resolveRef
   ---------->CRpsTree::rsvService
   <----------CRpsTree::rsvService (rc = 0)
   ---------->CRpsTree::rsvService
   <----------CRpsTree::rsvService (rc = 0)
   ---------->CRpsTree::rsvService
   <----------CRpsTree::rsvService (rc = 0)
   <--------CRpsTree::resolveRef (rc = 0)
   <------CRpsTree::refresh (rc = 0)
   !(00330) amUNLOCK_GLOBAL() 0
   <----amRpsOpen (rc = AM_ERR_OK)
   ---->amErrTranslate
   <----amErrTranslate (rc = 0)
   <--amSessCreateX (rc = 0)
 ...
```

Using trace (z/OS)

The AMI provides two types of trace on z/OS:

Formatted trace	Records spooled to a printer or directed to a file, which can be directly interpreted using TSO/ISPF browse, edit or print utilities.
GTF trace	Data captured on entry to and exit from high level and object level AMI function calls, which must be formatted by IPCS before viewing.

Formatted Trace

Formatted trace records are written on function entry and exit and at other points of execution where useful information can be gathered.

The format of the records is as follows:

```
Entry:
       hh:mm:ss.tttt ---->function name()

Exit:
       hh:mm:ss.tttt <----function name() (rc = n)

Data:
       ! information
```

Timestamps of entry and exit records are in local time, and are accurate to 1/10000 second. The function call depth is indicated for entry and exit records by the dashes in the '`---->`' or '`<----`' prefixes; two dashes per call level. For exit records, 'n' indicates the reason code on completion of the function. The default is to trace up to a depth of two function call levels, but this can be varied for batch applications. See "Control of formatted trace".

This a sample fragment from a formatted trace:

```
13:26:58.3263 -->amSendMsg
13:26:58.3264 ---->amSesGetSenderHandle
  ! amHashTableGetHandle failed.
13:26:58.3266 <----amSesGetSenderHandle (rc = [18][0x12])
13:26:58.3268 ---->amSesGetDistListHandle
  ! amHashTableGetHandle failed.
13:26:58.3269 <----amSesGetDistListHandle (rc = [18][0x12])
13:26:58.3270 ---->amSesCreateSender
```

For IMS, batch, or RRS-batch applications, formatted trace is directed to a dataset specified by the user. In the CICS environment, formatted trace entries are written to the current CICS trace destination as determined by the CICS administrator.

Control of formatted trace

For IMS, batch, or RRS-batch applications, formatted trace can be turned on by specifying a JCL 'DD' statement for DD name 'AMTTRACE'. This can be assigned to SYSOUT or to a DASD dataset. If assigned to SYSOUT, the trace records are written to a single spool file.

AMI formatted trace will not be started unless '//AMTTRACE DD' is specified.

If the trace dataset becomes full during an AMI session, the file will automatically be reopened and the trace will wrap.

For CICS applications, the AMI formatted trace is started if, at AMI session start, CICS internal and/or auxiliary trace is switched on. If the CICS trace destinations are stopped, AMI will perform no tracing for the session. The CICS administrator can use the CICS-supplied ‘CEMT’ transaction to control CICS trace.

For batch AMI applications, the trace level can be varied by specifying the Language Environment program parameter ‘ENVAR(AMT_TRACE_LEVEL=n)’. For example, to specify the formatted trace level for a C application program:

```
//JOBSTEP   EXEC PGM=AMIapp,PARM='ENVAR(AMT_TRACE_LEVEL=5)/'
```

For COBOL programs, Language Environment parameters are specified following the ‘/’ delimiter. For example:

```
//JOBSTEP   EXEC PGM=AMICob,PARM='/ENVAR(AMT_TRACE_LEVEL=9)'
```

Because CICS and IMS applications cannot easily set environment variables to control the trace level, the trace level defaults under CICS to a high setting. This ensures that all AMI trace points will be captured.

GTF Trace

AMI captures trace data for GTF at entry to and exit from each user-callable object level and high level AMI function. Entry trace data include function name and parameters. Exit trace data include function name and returned values.

IMS, batch, and RRS-batch AMI applications direct the trace data to GTF as user entries, using GTF event identifiers ‘5E9’ for entry, and ‘5EA’ for exit. These identifiers are the same as those used by WebSphere MQ for z/OS Application GTF trace, allowing for AMI and WebSphere MQ trace entries to be selected together in IPCS and formatted in a single, chronological, stream. Unlike WebSphere MQ, however, the GTF format identifier for AMI GTF trace records is ‘00’, causing IPCS to display these records in dump (hexadecimal/character) form, without using a bespoke formatting routine.

The following extract from IPCS formatted output shows an entry/exit pair of AMI GTF trace records:

```
HEXFORMAT AID FF FID 00 EID  E5E9
 +0000  00F63080  C1F8E2D5  C5D3D3E2  8194E285  ║ .6..A8SNELLSamSe ║
 +0010  A2C39385  8199C599  999699C3  968485A2  ║ sClearErrorCodes ║
 +0020  00000000  00000000  00000000  0FA05B10  ║ ................ ║
      GMT-11/05/1999 14:49:51.564812  LOC-11/05/1999 14:49:51.564812

HEXFORMAT AID FF FID 00 EID  E5EA
 +0000  00F63080  C1F8E2D5  C5D3D3E2  8194E285  ║ .6..A8SNELLSamSe ║
 +0010  A2C39385  8199C599  999699C3  968485A2  ║ sClearErrorCodes ║
 +0020  00000000  00000000  00000000  00000000  ║ ................ ║
 +0030  00000000                                ║ ....             ║
      GMT-11/05/1999 14:49:51.564906  LOC-11/05/1999 14:49:51.564906
```

AMI applications on CICS do not directly trace to GTF. AMI writes the same data to the current CICS trace destination(s) along with AMI formatted trace records. CICS tracing is controlled by the CICS administrator using the CICS-supplied transaction ‘CEMT’.

Control of GTF Trace

AMI writes GTF trace records if, at AMI session start, GTF is started for the application’s job name with option ‘TRACE=USR’. GTF is usually started from the z/OS operator’s console using an installation defined procedure. The chapter

“Using trace for problem determination” in the *WebSphere MQ for z/OS Problem Determination Guide* describes a typical GTF start-up prompt/reply sequence. If AMI and WebSphere MQ GTF trace entries are to be captured to the same dataset, the job names for both the AMI application and the WebSphere MQ queue manager must be specified.

If GTF is not started at the start of the AMI session, no GTF tracing will be performed for the remainder of the session.

When your AMI program fails

Reason Codes

When an AMI function call fails, it reports the level of the failure in the completion code of the call. AMI has three completion codes:

AMCC_OK	The call completed successfully
AMCC_WARNING	The call completed with unexpected results
AMCC_FAILED	An error occurred during processing

In the last two cases, AMI supplies a reason code that provides an explanation of the failure. A list of AMI reason codes is given in Appendix A, “Reason codes and LDAP error codes” on page 537.

In addition, if WebSphere MQ is the reason for the failure, AMI supplies a secondary reason code. The secondary reason codes can be found in the *WebSphere MQ Application Programming Reference* book.

First failure symptom report (iSeries, UNIX, and Windows)

A *first failure symptom* report is produced for unexpected and internal errors. This report is found in a file named `AMTnnnnn.FDC`, where `nnnnn` is the ID of the AMI process that is running at the time. You find this file in the working directory from which you started your AMI program, or in the directory specified by the path set in the AMT_TRACE_PATH environment variable. If you receive a first failure symptom report, you should contact IBM support personnel.

First failure symptom report (z/OS)

In the unlikely event that AMI detects an internal processing error from which no recovery is possible, the following actions are taken:

1. A dump is taken of the application’s data.
2. A first failure symptom report is produced.

Batch AMI applications write a Language Environment dump to SYSOUT. CICS AMI applications create a CICS transaction dump, with identifier ‘MAMT’.

Batch AMI applications write the first failure symptom report to the formatted trace data set (AMTTRACE), if allocated, otherwise to SYSOUT. CICS AMI applications write the symptom report to SYSOUT.

The formatted diagnostic information starts with a summary that includes:

```
Date/Time
Code Level
Function Name
Probe Id         (code point within function)
Build Date
Major Error Code
Minor Error Code
Comment Lines
```

Following the summary is a list of the stored function stack, indicating the current function call sequence. Following this is a list of the latest 40 function calls. Each item contains:

```
Entry/Exit indicator  Function name   Return Code
```

Other sources of information

AMI makes use of WebSphere MQ as a transport mechanism and so WebSphere MQ error logs and trace information can provide useful information. See the *WebSphere MQ System Administration* manual for details of how to activate these problem determination aids.

Common causes of problems

- With the C object interface, most functions require a handle to the object they refer to. If this handle is not valid, the results are unpredictable.
- Completion code 2 (AMRC_ERROR) together with reason code 110 (AMRC_TRANSPORT_NOT_AVAILABLE) returned by **amInitialize** or **amSesOpen** (or the equivalent in COBOL, C++ and Java) normally indicates that the underlying WebSphere MQ queue manager the AMI is attempting to use is not started (or does not exist). This might be because of a missing or incorrect xml repository file or because the data in the local host file is incorrect.
- Completion code 2 (AMRC_ERROR) together with reason code 49 (AMRC_TRANSPORT_ERR) indicates that an error was detected by the underlying WebSphere MQ transport. The secondary reason code returned by the appropriate ‘get last error’ function for the object concerned will provide the related WebSphere MQ reason code. This error occurs most frequently during an attempt to open an underlying WebSphere MQ queue object that does not exist (or has an incorrect type). This can be because it has never been created or because a missing or incorrect xml repository file is providing an incorrect queue name.

Part 7. Appendixes

Appendix A. Reason codes and LDAP error codes

This appendix contains a description of the AMRC_* reason codes, divided into three sections according to the value of the corresponding completion code. Within each section they are in alphabetic order. For a list of reason codes in numeric order, see Appendix B, “Constants and structures” on page 561.

In some circumstances, the AMI returns a secondary reason code that comes from WebSphere MQ, the underlying transport layer. Please refer to the *WebSphere MQ Application Programming Reference* manual for details of these reason codes.

This appendix also contains a description of the secondary error code values that can occur when you update an LDAP directory with AMI information from a repository file. These values might be displayed in an AMI Administration Tool error message, or returned from a GetLastError request for the AMI object concerned.

Reason code: OK

The following reason code is returned with completion code: AMCC_OK

AMRC_NONE
The request was successful with no error or warning returned.

Reason code: Warning

The following reason codes are returned with completion code: AMCC_WARNING

AMRC_BACKED_OUT
The unit of work has been backed out.

AMRC_BACKOUT_LIMIT_ERR
The backout count of a received message was found to have exceeded its backout limit. The message was returned to the application and was requeued to the backout-requeue queue.

AMRC_BACKOUT_REQUEUE_ERR
The backout count of a received message was found to have exceeded its backout limit. The message was returned to the application. It could not be requeued to the backout requeue queue.

AMRC_CCSID_NOT_SUPPORTED
OS/390 V2 R9 or later is required to enable AMI publish/subscribe or message element support under CICS. Ensure that your Language Environment installation is set up to use Unicode character conversion. See the *OS/390 C/C++ Programming Guide* for a list of the coded character sets supported under z/OS.

AMRC_CLOSE_SESSION_ERR
An error occurred while closing the session. The session is closed.

AMRC_ENCODING_INCOMPLETE
The message contains mixed values for integer, decimal, and floating point encodings, one or more of which are undefined. The encoding value returned to the application reflects only the encoding values that were defined.

Reason code (warning)

AMRC_ENCODING_MIXED
The message contains mixed values for integer, decimal and floating point encodings, one or more of which conflict. An encoding value of undefined was returned to the application.

AMRC_FILE_ALREADY_EXISTS
The AMI was unable to receive the file as the current file disposition is ‘new’, and a file with the same name already exists on your system. The first message of the file transfer is returned to the application. If this occurs, we recommend that the current unit of work is backed out. This will ensure that the messages received from the service are in a consistent state.

AMRC_FILE_FORMAT_CONVERTED
The AMI received a file successfully, but needed to convert between different file types. An example is from an z/OS fixed-length dataset to a UNIX file or between z/OS datasets with different geometries.

AMRC_FILE_NOT_WRITTEN
The file used for a receive could not be opened. The first message of the file is returned to the application. If this occurs we recommend that the current unit of work is backed out. This will ensure that the messages held on the service are in a consistent state.

AMRC_FILE_SYSTEM_ERR
A filesystem error occurred during a file transfer call. If this occurs, we recommend that the current unit of work is backed out. This will ensure the messages put to or received from the service are in a consistent state.

AMRC_FILE_TRUNCATED
On a file send or receive operation, the entire file was not processed. We recommend that the current unit of work is backed out. This will ensure that the messages put to or received from the service are in a consistent state.

AMRC_GROUP_BACKOUT_LIMIT_ERR
The backout count of a received message was found to have exceeded its backout limit. The message was returned to the application. It was not requeued to the backout requeue queue because it represented a single message within a group of more than one.

AMRC_GROUP_MSG_RETURNED
A group message has been retrieved from a queue that does not have INDXTYPE(GROUPID). The group sequence is therefore not preserved. If you want the group messages to be retrieved in sequence, you must alter the INDXTYPE of the queue to GROUPID. This warning occurs on z/OS only.

AMRC_MULTIPLE_REASONS
A distribution list open or send was only partially successful and returned multiple different reason codes in its underlying sender services.

AMRC_MSG_TRUNCATED
The received message that was returned to the application has been truncated.

AMRC_NO_REPLY_TO_INFO
A response sender service specified when attempting to receive a request message was not updated with reply-to information because the request message contained no reply-to information. An attempt to send a reply message using the response sender will fail.

AMRC_NON_GROUP_MSG_IN_GROUP
The message was sent succesfully, but an incomplete message group has been sent and the message was not the next message in the group. The information

about the incomplete group is no longer stored so the group cannot be continued. If the group is meant to be complete, ensure that the last message sent in the group specifies AMGRP_LAST_MSG_IN_GROUP.

AMRC_NOT_A_FILE
A message was received from the service, but it does not appear to have been sent as part of a (physical mode) file transfer operation. The message is returned to the application.

AMRC_NOT_CONVERTED
Data conversion of the received message was unsuccessful. The message was removed from the underlying message transport layer with the message data unconverted.

AMRC_POLICY_HANDLER_WARNING
A warning was returned from a policy handler library invocation that occurred while processing the application function call. The policy handler reason code can be obtained by the secondary reason code value returned from a getlastError request for the AMI object concerned.

AMRC_POLICY_NOT_IN_REPOS
The definition name that was specified when creating a policy was not found in the repository. The policy was created using default values.

AMRC_PUBLISHER_NOT_IN_REPOS
The definition name that was specified when creating a publisher was not found in the specified repository. The publisher was created using default values.

AMRC_RECEIVER_NOT_IN_REPOS
The definition name that was specified when creating a receiver was not found in the repository. The receiver was created using default values.

AMRC_REPOS_WARNING
A warning associated with the underlying repository data was reported.

AMRC_RFH2_FORMAT_ERR
The format of an MQRFH2 rules and formatting header of a received message was not valid.

AMRC_SENDER_NOT_IN_REPOS
The definition name that was specified when creating a sender was not found in the repository. The sender was created using default values.

AMRC_SUBSCRIBER_NOT_IN_REPOS
The definition name that was specified when creating a subscriber was not found in the repository. The subscriber was created using default values.

AMRC_TRANSPORT_WARNING
A warning was reported by the underlying (WebSphere MQ) message transport layer. The message transport reason code can be obtained by the secondary reason code value returned from a 'GetLastError' request for the AMI object concerned.

AMRC_UNEXPECTED_RECEIVE_ERR
An unexpected error occurred after a received message was removed from the underlying transport layer. The message was returned to the application.

AMRC_UNEXPECTED_SEND_ERR
An unexpected error occurred after a message was successfully sent. Output information updated as a result of the send request should never occur.

Reason code: Failed

The following reason codes are returned with completion code: AMCC_FAILED

AMRC_BACKOUT_INVALID
The backout request was not valid. On z/OS under CICS, IMS, or RRS this can be due to calling the AMI backout functions rather than the transaction managers' own functions.

AMRC_BEGIN_INVALID
The begin request was not valid because there were no participating resource managers registered.

AMRC_BROWSE_OPTIONS_ERR
The specified browse options value was not valid or contained a combination of options that is not valid.

AMRC_CCSID_ERR
The specified coded character value was not valid.

AMRC_CCSID_NOT_SUPPORTED
The coded character set of name/value elements in the rules and formatting header of a received message, or that specified for passing elements between the application and the AMI, is not supported.

AMRC_CCSID_PTR_ERR
The specified coded character set id pointer was not valid.

AMRC_COMMAND_ALREADY_EXISTS
A publish, subscribe, or unsubscribe command could not be added to the message because the message already contained a command element. If this message is generated from the high-level interface, it may mean that you have tried to use the same message name for sending and receiving publish/subscribe messages. It can also occur if the same message object is reused to send a message without being reset.

AMRC_COMMIT_INVALID
The commit request was not valid. On z/OS under CICS, IMS, or RRS this can be due to calling the AMI commit functions rather than the transaction managers' own functions.

AMRC_CONN_NAME_NOT_FOUND
The connection name obtained from the repository was not found in the local host file.

AMRC_CORREL_ID_BUFF_LEN_ERR
The specified correlation id buffer length value was not valid.

AMRC_CORREL_ID_BUFF_PTR_ERR
The specified correlation id buffer pointer was not valid.

AMRC_CORREL_ID_LEN_ERR
The specified correlation id length value was too long.

AMRC_CORREL_ID_LEN_PTR_ERR
The specified correlation id length pointer was not valid.

AMRC_CORREL_ID_PTR_ERR
The specified correlation id pointer was not valid.

AMRC_DATA_BUFF_LEN_ERR
The specified data buffer length value was not valid.

AMRC_DATA_BUFF_PTR_ERR
The specified data buffer pointer was not valid.

AMRC_DATA_LEN_ERR
The specified data length was not valid.

AMRC_DATA_LEN_PTR_ERR
The specified data length pointer was not valid.

AMRC_DATA_OFFSET_ERR
The specified data offset value was not valid.

AMRC_DATA_OFFSET_PTR_ERR
The specified data offset pointer was not valid.

AMRC_DATA_PTR_ERR
The specified data pointer was not valid.

AMRC_DATA_SOURCE_NOT_UNIQUE
Message data for a send operation was passed in an application data buffer or a file, and was also found in the specified message object. Data to be sent can be included in an application buffer or a message object, but not both. Similarly, data can be included in a file or a message object, but not both. If data is sent in an application buffer or file, the message object can be reset first to remove existing data.

AMRC_DEFN_TYPE_ERR
The definition type defined for the service point in the repository was inconsistent with the definition type of the underlying message transport queue object when it was opened.

AMRC_DEFN_TYPE_PTR_ERR
The specified definition type pointer was not valid.

AMRC_DIST_LIST_INDEX_ERR
The specified distribution list index value was not valid.

AMRC_DIST_LIST_NOT_IN_REPOS
The definition name specified for creating a distribution list was not found in the repository. The object was not created.

AMRC_DTD_NOT_FOUND
An AMI dtd file (amt.dtd) was not found with the xml repository file in the same directory.

AMRC_DIST_LIST_NOT_UNIQUE
The specified name could not be resolved to a unique distribution list because more than one distribution list with that name exists.

AMRC_ELEM_COUNT_PTR_ERR
The specified element count pointer was not valid.

AMRC_ELEM_INDEX_ERR
The specified element index value was not valid.

AMRC_ELEM_NAME_LEN_ERR
The specified element name length value was not valid.

AMRC_ELEM_NAME_PTR_ERR
The specified element name pointer was not valid.

AMRC_ELEM_NOT_FOUND
The specified element was not found.

Reason code (failed)

AMRC_ELEM_PTR_ERR
The specified element pointer was not valid.

AMRC_ELEM_STRUC_ERR
The specified element structure was not valid. The structure id, version, or a reserved field contained a value that is not valid.

AMRC_ELEM_STRUC_NAME_BUFF_ERR
At least one of the name buffer (length and pointer) fields in the specified element structure was not valid.

AMRC_ELEM_STRUC_NAME_ERR
At least one of the name (length and pointer) fields in the specified element structure was not valid. Ensure that the name length, pointer, and name string are valid.

AMRC_ELEM_STRUC_TYPE_BUFF_ERR
At least one of the type buffer (length and pointer) fields in the specified element structure was not valid. Ensure that the type length, pointer and type string are valid.

AMRC_ELEM_STRUC_TYPE_ERR
At least one of the type (length and pointer) fields in the specified element structure was not valid.

AMRC_ELEM_STRUC_VALUE_BUFF_ERR
At least one of the value buffer (length and pointer) fields in the specified structure was not valid.

AMRC_ELEM_STRUC_VALUE_ERR
At least one of the value (length and pointer) fields in the specified element structure was not valid. Ensure that the value length, pointer, and value string are valid.

AMRC_ENCODING_ERR
The specified encoding value was not valid.

AMRC_ENCODING_PTR_ERR
The specified encoding pointer was not valid.

AMRC_FILE_FORMAT_NOT_SUPPORTED
An attempt was made to send a file type that is not supported. Unsupported file types include z/OS VSAM datasets, and z/OS partitioned datasets (though an individual member of a PDS may be sent).

AMRC_FILE_MSG_FORMAT_ERR
When using physical mode file transfer, only two message formats are allowed: AMFMT_STRING (for text mode transfer), and AMFMT_NONE (for binary mode transfer). When using logical mode file transfer, any message format may be used for messages generated from z/OS datasets. On other platforms and for HFS files on z/OS, only AMFMT_STRING and AMFMT_NONE can be used.

AMRC_FILE_NAME_LEN_ERR
The file name length passed in to a file transfer call was not valid.

AMRC_FILE_NAME_PTR_ERR
The file name pointer passed in to a file transfer call was not valid.

AMRC_FILE_NOT_FOUND
The file supplied on a file send call could not be opened. Check that the file exists and that the application has read access to it.

AMRC_FILE_TRANSFER_INVALID
An application running under CICS on z/OS tried to perform a file transfer operation that is not valid in this environment.

AMRC_FORMAT_BUFF_LEN_ERR
The specified format buffer length value was not valid.

AMRC_FORMAT_BUFF_PTR_ERR
The specified format buffer pointer was not valid.

AMRC_FORMAT_LEN_ERR
The specified message format string was too long.

AMRC_FORMAT_LEN_PTR_ERR
The specified format length pointer was not valid.

AMRC_FORMAT_PTR_ERR
The specified format pointer was not valid.

AMRC_GROUP_STATUS_ERR
The specified group status value was not valid.

AMRC_GROUP_STATUS_PTR_ERR
The specified group status pointer was not valid.

AMRC_HEADER_INVALID
The RFH header structure of the message was not valid.

AMRC_HEADER_TRUNCATED
The RFH header of the message was truncated.

AMRC_HOST_CACHE_ERR
A module was loaded for use as a repository file cache, but the module does not appear to be a valid repository cache.

AMRC_HOST_FILE_ERR
The contents of the local host file are not valid.

AMRC_HOST_FILENAME_ERR
The local host file name was not valid. The value of the appropriate environment variable should be corrected.

AMRC_HOST_FILE_NOT_FOUND
A local host file with the specified name was not found.

AMRC_INCOMPLETE_GROUP
The specified request failed because an attempt was made to send a message that was not in a group when the existing message group was incomplete.

AMRC_INSUFFICIENT_MEMORY
There was not enough memory available to complete the requested operation.

AMRC_INVALID_DIST_LIST_NAME
The specified distribution list name was too long, contained characters that are not valid, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_IF SERVICE_OPEN
The requested operation could not be performed because the specified service (sender, receiver, publisher, or subscriber) was open.

AMRC_INVALID_MSG_NAME
The specified message name was too long, contained characters that are not valid, or used the reserved prefix 'SYSTEM.'.

Reason code (failed)

AMRC_INVALID_POLICY_NAME
The specified policy name was too long, contained characters that are not valid, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_PUBLISHER_NAME
The specified publisher service name was too long, contained characters that are not valid, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_Q_NAME
The specified queue name was too long, or contained characters that are not valid.

AMRC_INVALID_RECEIVER_NAME
The specified receiver service name was too long, contained characters that are not valid, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_SENDER_NAME
The specified sender service name was too long, contained characters that are not valid, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_SESSION_NAME
The specified session name was too long, contained characters that are not valid, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_SUBSCRIBER_NAME
The specified subscriber service name was too long, contained characters that are not valid, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_TRACE_LEVEL
A specified trace level was not valid.

AMRC_JAVA_CLASS_ERR
A class referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_CREATE_ERR
An unexpected error occurred when creating an AMI Java object. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_FIELD_ERR
A field referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_JNI_ERR
An unexpected error occurred when calling the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_METHOD_ERR
A method referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_LDAP_ERR
An error was encountered accessing the AMI repository information in the LDAP directory, or communicating with the LDAP server. The LDAP error code can be obtained from the secondary reason code value that is returned from a GetLastError request for the AMI object concerned. See "LDAP error codes" on page 556.

AMRC_LIBRARY_DUP_FUNCTION
A policy handler library that is specified by the repository attempted to register a function with an invocation point value that it has already registered.

AMRC_LIBRARY_FUNCTION_PTR_ERR
A policy handler library that is specified by the repository attempted to register a function with a function pointer value that is not valid (for example, NULL).

AMRC_LIBRARY_INV_POINT_ERR
A policy handler library that is specified by the repository attempted to register a function with an invocation point value that was not valid.

AMRC_LIBRARY_NOT_FOUND
A policy handler library file name specified in the repository was not found in the `handlers` directory.

AMRC_JAVA_NULL_PARM_ERR
The AMI Java code detected a null parameter that is not valid. (Not applicable to the C and C++ programming languages).

AMRC_MSG_HANDLE_ERR
The specified message handle was not valid.

AMRC_MSG_ID_BUFF_LEN_ERR
The specified message id buffer length value was not valid.

AMRC_MSG_ID_BUFF_PTR_ERR
The specified message id buffer pointer was not valid.

AMRC_MSG_ID_LEN_ERR
The specified message id length value was not valid.

AMRC_MSG_ID_LEN_PTR_ERR
The specified message id length pointer was not valid.

AMRC_MSG_ID_PTR_ERR
The specified message id pointer was not valid.

AMRC_MSG_NOT_FOUND
The specified message was not found, so the request was not carried out.

AMRC_MSG_NOT_UNIQUE
The specified name could not be resolved to a unique message because more than one message object with that name exists.

AMRC_MSG_TYPE_NOT_REPORT
The message is not a report message.

AMRC_MSG_TYPE_PTR_ERR
The specified message type pointer was not valid.

AMRC_NAME_BUFF_LEN_ERR
The specified name buffer length value was not valid.

AMRC_NAME_BUFF_PTR_ERR
The specified name buffer pointer was not valid.

Reason code (failed)

AMRC_NAME_LEN_PTR_ERR
The specified name length pointer was not valid.

AMRC_NEGATIVE_RECEIVE_BUFF_LEN
The value of the buffer length parameter that is specified on a receive message request was negative.

AMRC_NO_MSG_AVAILABLE
No message was available for a receive request after the specified wait time.

AMRC_NO_RESP_SERVICE
The publish request was not successful because a response receiver service is required for registration and was not specified.

AMRC_NOT_AUTHORIZED
The user is not authorized by the underlying transport layer to perform the specified request.

| **AMRC_POLICY_HANDLER_ERROR**
| An error was returned from a policy handler library invocation that occurred
| while processing the application function call. The policy handler reason code
| can be obtained by the secondary reason code value returned from a
| getlastError request for the AMI object concerned.

AMRC_POLICY_NOT_FOUND
The specified policy was not found, so the request was not carried out.

AMRC_POLICY_NOT_UNIQUE
The specified name could not be resolved to a unique policy because more than one policy with that name exists.

AMRC_PRIMARY_HANDLE_ERR
The primary handle (that is, the first parameter) passed on the API call was not valid. The most probable reason for failure is that the handle passed is a synonym handle, which is not valid as the *primary* handle on any call to the AMI.

AMRC_PUBLISHER_NOT_UNIQUE
The specified name could not be resolved to a unique publisher because more than one publisher object with that name exists.

AMRC_Q_NAME_BUFF_LEN_ERR
The specified queue name buffer length value was not valid.

AMRC_Q_NAME_BUFF_PTR_ERR
The specified queue name buffer pointer was not valid.

AMRC_Q_NAME_LEN_ERR
The specified queue name length value was not valid.

AMRC_Q_NAME_LEN_PTR_ERR
The specified queue name length pointer was not valid.

AMRC_Q_NAME_PTR_ERR
The specified queue name pointer was not valid.

AMRC_READ_OFFSET_ERR
The current data offset used for reading bytes from a message is not valid.

AMRC_RECEIVE_BUFF_LEN_ERR
The buffer length specified for receiving data was not valid.

AMRC_RECEIVE_BUFF_PTR_ERR
The buffer pointer specified for receiving data was not valid.

AMRC_RECEIVE_DISABLED
The specified request could not be performed because the service in the underlying transport layer is not enabled for receive requests.

AMRC_RECEIVER_NOT_UNIQUE
The specified name could not be resolved to a unique receiver because more than one receiver object with that name exists.

AMRC_REPORT_CODE_ERR
The specified report (or feedback) code value was not valid.

AMRC_REPORT_CODE_PTR_ERR
The specified report code pointer was not valid.

AMRC_REPOS_CACHE_ERR
A module was loaded for use as a host file cache, but the module does not appear to be a valid host cache.

AMRC_REPOS_ERR
An error was returned when initializing or accessing the repository. This can occur for any of the following reasons:

- The repository XML file (for instance, `amt.xml`) contains data that is not valid.
- The DTD file (`amt.dtd`) was not found or contains data that is not valid.
- The files needed to initialize the repository (located in directories `intlFiles` and `locales`) could not be located.

Check that the DTD and XML files are valid and correctly located, and that the path settings for the local host and repository files are correct.

AMRC_REPOS_FILENAME_ERR
The repository file name was not valid. The value of the appropriate environment variable should be corrected.

AMRC_REPOS_NOT_FOUND
The repository file was not found. The value of the appropriate environment variable should be corrected.

AMRC_RESERVED_NAME_IN_REPOS
The name specified for creating an object was found in the repository and is a reserved name that is not valid in a repository. The specified object was not created.

AMRC_RESP_RECEIVER_HANDLE_ERR
The response receiver service handle specified when sending a request message was not valid.

AMRC_RESP_SENDER_HANDLE_ERR
The response sender service handle specified when receiving a request message was not valid.

AMRC_RFH_ALREADY_EXISTS
A publish, subscribe, or unsubscribe command could not be added to the message because the message already contained an RFH header. The message requires a reset first, to remove existing data.

AMRC_SEND_DATA_PTR_ERR
The buffer pointer specified for sending data was not valid.

AMRC_SEND_DATA_LEN_ERR
The data length specified for sending data was not valid.

Reason code (failed)

AMRC_SEND_DISABLED
The specified request could not be performed because the service in the underlying transport layer is not enabled for send requests.

AMRC_SENDER_COUNT_PTR_ERR
The specified distribution list sender count pointer was not valid.

AMRC_SENDER_NOT_UNIQUE
The specified name could not be resolved to a unique sender because more than one sender object with that name exists.

AMRC_SENDER_USAGE_ERR
The specified sender service definition type was not valid for sending responses. To be valid for sending a response, a sender service must not have a repository definition, must have been specified as a response service when receiving a previous request message and must not have been used for any purpose other than sending responses.

AMRC_SERVICE_ALREADY_CLOSED
The specified (sender, receiver, distribution list, publisher or subscriber) service was already closed.

AMRC_SERVICE_ALREADY_OPEN
The specified (sender, receiver, distribution list, publisher or subscriber) service was already open.

AMRC_SERVICE_FULL
The specified request could not be performed because the service in the underlying transport has reached its maximum message limit.

AMRC_SERVICE_HANDLE_ERR
The service handle specified for a sender, receiver, distribution list, publisher, or subscriber was not valid.

AMRC_SERVICE_NOT_FOUND
The specified (sender, receiver, distribution list, publisher, or subscriber) service was not found, so the request was not carried out.

AMRC_SERVICE_NOT_OPEN
The request failed because the specified (sender, receiver, distribution list, publisher or subscriber) service was not open.

AMRC_SESSION_ALREADY_CLOSED
The session was already closed (or terminated).

AMRC_SESSION_ALREADY_OPEN
The session was already open (or initialized).

AMRC_SESSION_EXPIRED
Under the IMS environment, the current session has been marked as expired. See "IMS applications" on page 27 for an explanation of why a session may be expired. Delete the current session and create new one for the duration of this transaction.

AMRC_SESSION_HANDLE_ERR
The specified session handle was not valid.

AMRC_SESSION_NOT_OPEN
The request failed because the session was not open.

AMRC_SUBSCRIBER_NOT_UNIQUE
The specified name could not be resolved to a unique subscriber because more than one subscriber object with that name exists.

AMRC_TRANSPORT_ERR
An error was reported by the underlying (WebSphere MQ) message transport layer. The message transport reason code can be obtained by the secondary reason code value returned from a 'GetLastError' request for the AMI object concerned. For more information, see "Common causes of problems" on page 533.

AMRC_TRANSPORT_LIBRARY_ERR
An error occurred loading the transport library.

AMRC_TRANSPORT_NOT_AVAILABLE
The underlying transport layer is not available.

AMRC_UNEXPECTED_ERR
An unexpected error occurred.

AMRC_WAIT_TIME_ERR
The specified wait-time value was not valid.

AMRC_WAIT_TIME_PTR_ERR
The specified wait time pointer was not valid.

AMRC_WAIT_TIME_READ_ONLY
An attempt was made to set the wait time in a policy object for which the wait-time was read-only.

Reason Code: Failed (Extended C AMI functions)

The following reason codes are returned with completion code: AMCC_FAILED

They are returned only by the extended C AMI functions.

AMRC_ACCEPT_DIRECT_ERR
The specified accept direct requests value was not valid.

AMRC_ACCEPT_DIRECT_PTR_ERR
The specified accept direct requests pointer was not valid.

AMRC_ACCEPT_TRUNCATED_ERR
The specified accept truncated value was not valid.

AMRC_ACCEPT_TRUNCATED_PTR_ERR
The specified accept truncated pointer was not valid.

AMRC_ANON_ERR
The specified anonymous value was not valid.

AMRC_ANON_PTR_ERR
The specified anonymous pointer was not valid.

AMRC_APPL_GROUP_BUFF_LEN_ERR
The specified application group buffer length value was not valid.

AMRC_APPL_GROUP_BUFF_PTR_ERR
The specified application group buffer pointer was not valid.

AMRC_APPL_GROUP_LEN_ERR
The specified application group length value was not valid.

AMRC_APPL_GROUP_LEN_PTR_ERR
The specified application group length pointer was not valid.

AMRC_APPL_GROUP_PTR_ERR
The specified application group pointer was not valid.

Reason code (failed)

AMRC_BIND_ON_OPEN_ERR
The specified bind on open value was not valid.

AMRC_BIND_ON_OPEN_PTR_ERR
The specified bind on open pointer was not valid.

AMRC_CHL_NAME_BUFF_LEN_ERR
The specified channel name buffer length value was not valid.

AMRC_CHL_NAME_BUFF_PTR_ERR
The specified channel name buffer pointer was not valid.

AMRC_CHL_NAME_LEN_ERR
The specified channel name length value was not valid.

AMRC_CHL_NAME_LEN_PTR_ERR
The specified channel name length pointer was not valid.

AMRC_CHL_NAME_PTR_ERR
The specified channel name pointer was not valid.

AMRC_CLOSE_DELETE_ERR
The specified close delete value was not valid.

AMRC_CLOSE_DELETE_PTR_ERR
The specified close delete pointer was not valid.

AMRC_CON_HANDLE_ERR
The specified connection handle was not valid.

AMRC_CON_INT_PROP_ID_ERR
The specified connection integer property identifier was not valid.

AMRC_CON_STR_PROP_ID_ERR
The specified connection string property identifier was not valid.

AMRC_CONTEXT_ERR
The specified message context value was not valid.

AMRC_CONTEXT_PTR_ERR
The specified message context pointer was not valid.

AMRC_CONVERT_ERR
The specified convert message value was not valid.

AMRC_CONVERT_PTR_ERR
The specified convert message pointer was not valid.

AMRC_COUNT_ERR
The specified backout or retry count value was not valid.

AMRC_COUNT_PTR_ERR
The specified backout or retry count pointer was not valid.

AMRC_CUST_PARM_BUFF_LEN_ERR
The specified custom parameter buffer length value was not valid.

AMRC_CUST_PARM_BUFF_PTR_ERR
The specified custom parameter buffer pointer was not valid.

AMRC_CUST_PARM_LEN_ERR
The specified custom parameter length value was not valid.

AMRC_CUST_PARM_LEN_PTR_ERR
The specified custom parameter length pointer was not valid.

AMRC_CUST_PARM_PTR_ERR
The specified custom parameter pointer was not valid.

AMRC_DLY_PERSISTENCE_ERR
The specified delivery persistence value was not valid.

AMRC_DLY_PERSISTENCE_PTR_ERR
The specified delivery persistence pointer was not valid.

AMRC_DST_SUPPORT_ERR
The specified distribution list support value was not valid.

AMRC_DST_SUPPORT_PTR_ERR
The specified distribution list support pointer was not valid.

AMRC_EXPIRY_ERR
The specified message expiry value was not valid.

AMRC_EXPIRY_PTR_ERR
The specified message expiry pointer was not valid.

AMRC_FILE_DISP_ERR
The specified file disposition value was not valid.

AMRC_FILE_DISP_PTR_ERR
The specified file disposition pointer was not valid.

AMRC_FILE_RCD_LEN_ERR
The specified file record length value was not valid.

AMRC_FILE_RCD_LEN_PTR_ERR
The specified file record length pointer was not valid.

AMRC_GROUP_ID_BUFF_LEN_ERR
The specified group id group buffer length value was not valid.

AMRC_GROUP_ID_BUFF_PTR_ERR
The specified group id buffer pointer was not valid.

AMRC_GROUP_ID_LEN_ERR
The specified group id length value was not valid.

AMRC_GROUP_ID_LEN_PTR_ERR
The specified group id length pointer was not valid.

AMRC_GROUP_ID_PTR_ERR
The specified group id pointer was not valid.

AMRC_HANDLE_POISON_MSG_ERR
The specified handle poison message value was not valid.

AMRC_HANDLE_POISON_MSG_PTR_ERR
The specified handle poison message pointer was not valid.

AMRC_HANDLE_PTR_ERR
The specified handle pointer was not valid.

AMRC_IMPL_OPEN_ERR
The specified implicit open value was not valid.

AMRC_IMPL_OPEN_PTR_ERR
The specified implicit open pointer was not valid.

AMRC_INFORM_IF_RET_ERR
The specified inform if retained value was not valid.

Reason code (failed)

AMRC_INFORM_IF_RET_PTR_ERR
The specified inform if retained pointer was not valid.

AMRC_INTERVAL_ERR
The specified retry interval value was not valid.

AMRC_INTERVAL_PTR_ERR
The specified retry interval pointer was not valid.

AMRC_INVALID_IF_CON_OPEN
The requested operation could not be performed because the specified connection was open.

AMRC_LEAVE_OPEN_ERR
The specified leave open value was not valid.

AMRC_LEAVE_OPEN_PTR_ERR
The specified leave open pointer was not valid.

AMRC_LOCAL_ERR
The specified publish or subscribe locally value was not valid.

AMRC_LOCAL_PTR_ERR
The specified publish or subscribe locally pointer was not valid.

AMRC_MCD_PARM_BUFF_LEN_ERR
The specified MCD parameter buffer length value was not valid.

AMRC_MCD_PARM_BUFF_PTR_ERR
The specified MCD parameter buffer pointer was not valid.

AMRC_MCD_PARM_LEN_ERR
The specified MCD parameter length value was not valid.

AMRC_MCD_PARM_LEN_PTR_ERR
The specified MCD parameter length pointer was not valid.

AMRC_MCD_PARM_PTR_ERR
The specified MCD parameter pointer was not valid.

AMRC_MGR_NAME_BUFF_LEN_ERR
The specified queue manager name buffer length value was not valid.

AMRC_MGR_NAME_BUFF_PTR_ERR
The specified queue manager name buffer pointer was not valid.

AMRC_MGR_NAME_LEN_ERR
The specified queue manager name length value was not valid.

AMRC_MGR_NAME_LEN_PTR_ERR
The specified queue manager name length pointer was not valid.

AMRC_MGR_NAME_PTR_ERR
The specified queue manager name pointer was not valid.

AMRC_MSG_INT_PROP_ID_ERR
The specified message integer property identifier was not valid.

AMRC_MSG_LEN_ERR
The specified message length value was not valid.

AMRC_MSG_LEN_PTR_ERR
The specified message length pointer was not valid.

AMRC_MSG_STR_PROP_ID_ERR
The specified message string property identifier was not valid.

AMRC_MSG_TYPE_ERR
The specified message type value was not valid.

AMRC_NEW_CORREL_ID_ERR
The specified new correlation id value was not valid.

AMRC_NEW_CORREL_ID_PTR_ERR
The specified new correlation id pointer was not valid.

AMRC_NEW_PUBS_ONLY_ERR
The specified new publications only value was not valid.

AMRC_NEW_PUBS_ONLY_PTR_ERR
The specified new publications only pointer was not valid.

AMRC_PERSISTENCE_ERR
The specified persistence value was not valid.

AMRC_PERSISTENCE_PTR_ERR
The specified persistence pointer was not valid.

AMRC_POLICY_INT_PROP_ID_ERR
The specified policy integer property identifier was not valid.

AMRC_POLICY_STR_PROP_ID_ERR
The specified policy string property identifier was not valid.

AMRC_PRIORITY_ERR
The specified priority value was not valid.

AMRC_PRIORITY_PTR_ERR
The specified priority pointer was not valid.

AMRC_PUB_ON_REQ_ERR
The specified publish on request value was not valid.

AMRC_PUB_ON_REQ_PTR_ERR
The specified publish on request pointer was not valid.

AMRC_PUB_OTHERS_ONLY_ERR
The specified publish to others only value was not valid.

AMRC_PUB_OTHERS_ONLY_PTR_ERR
The specified publish to others only pointer was not valid.

AMRC_READ_ONLY_ERR
The specified wait time read only value was not valid.

AMRC_READ_ONLY_PTR_ERR
The specified wait time read only pointer was not valid.

AMRC_REMOVE_ALL_ERR
The specified remove all subscriptions value was not valid.

AMRC_REMOVE_ALL_PTR_ERR
The specified remove all subscriptions pointer was not valid.

AMRC_REPORT_OPTION_ERR
The specified report option value was not valid.

AMRC_REPORT_OPTION_PTR_ERR
The specified report option pointer was not valid.

AMRC_RETAIN_ERR
The specified retain publications value was not valid.

Reason code (failed)

AMRC_RETAIN_PTR_ERR
The specified retain publications pointer was not valid.

AMRC_SEGMENT_ERR
The specified segment message value was not valid.

AMRC_SEGMENT_PTR_ERR
The specified segment message pointer was not valid.

AMRC_SEQ_NO_ERR
The specified sequence number value was not valid.

AMRC_SEQ_NO_PTR_ERR
The specified sequence number pointer was not valid.

AMRC_SET_NAME_INVALID
The specified name cannot be changed.

AMRC_SHARED_ERR
The specified open shared value was not valid.

AMRC_SHARED_PTR_ERR
The specified open shared pointer was not valid.

AMRC_SND_TYPE_ERR
The specified sender type value was not valid.

AMRC_SND_TYPE_PTR_ERR
The specified sender type pointer was not valid.

AMRC_SPLIT_LOGICAL_ERR
The specified split logical value was not valid.

AMRC_SPLIT_LOGICAL_PTR_ERR
The specified split logical pointer was not valid.

AMRC_SRV_INT_PROP_ID_ERR
The specified service integer property identifier was not valid.

AMRC_SRV_STR_PROP_ID_ERR
The specified service string property identifier was not valid.

AMRC_SRV_TYPE_ERR
The specified service type value was not valid.

AMRC_SRV_TYPE_PTR_ERR
The specified service type pointer was not valid.

AMRC_SUBS_POINT_BUFF_LEN_ERR
The specified subscription point buffer length value was not valid.

AMRC_SUBS_POINT_BUFF_PTR_ERR
The specified subscription point buffer pointer was not valid.

AMRC_SUBS_POINT_LEN_ERR
The specified subscription point length value was not valid.

AMRC_SUBS_POINT_LEN_PTR_ERR
The specified subscription point length pointer was not valid.

AMRC_SUBS_POINT_PTR_ERR
The specified subscription point pointer was not valid.

AMRC_SUPPRESS_REG_ERR
The specified suppress registration value was not valid.

AMRC_SUPPRESS_REG_PTR_ERR
The specified suppress registration pointer was not valid.

AMRC_SYNCPOINT_ERR
The specified sync point value was not valid.

AMRC_SYNCPOINT_PTR_ERR
The specified sync point pointer was not valid.

AMRC_TCP_ADDR_BUFF_LEN_ERR
The specified TCP/IP address buffer length value was not valid.

AMRC_TCP_ADDR_BUFF_PTR_ERR
The specified TCP/IP address buffer pointer was not valid.

AMRC_TCP_ADDR_LEN_ERR
The specified TCP/IP address length value was not valid.

AMRC_TCP_ADDR_LEN_PTR_ERR
The specified TCP/IP address length pointer was not valid.

AMRC_TCP_ADDR_PTR_ERR
The specified TCP/IP address pointer was not valid.

AMRC_TRP_TYPE_ERR
The specified transport type value was not valid.

AMRC_TRP_TYPE_PTR_ERR
The specified transport type pointer was not valid.

AMRC_TRUSTED_ERR
The specified trusted value was not valid.

AMRC_TRUSTED_PTR_ERR
The specified trusted pointer was not valid.

AMRC_USE_CORREL_ID_ERR
The specified use correlation id value was not valid.

AMRC_USE_CORREL_ID_PTR_ERR
The specified use correlation id pointer was not valid.

AMRC_WAIT_WHOLE_GROUP_ERR
The specified wait for whole group value was not valid.

AMRC_WAIT_WHOLE_GROUP_PTR_ERR
The specified wait for whole group pointer was not valid.

LDAP error codes

1 Unable to allocate memory

Cause:	Not enough memory available to perform the update.
Action:	Close any other applications that are running on the workstation.

2 Unable to access or read SSL keyfile

Cause:	The Secure Sockets Layer key file cannot be located.
Action:	If you require SSL encryption, check the name of the file is specified correctly.

3 Invalid SSL keyfile password

Cause:	The password for the Secure Sockets Layer key file is incorrect.
Action:	Correct the keyfile password.

4 Invalid SSL key name

Cause:	The private key name to use in the Secure Sockets Layer key file is incorrect.
Action:	Ensure that the key name specified is correct. The key name is the name or label associated with the private key/certificate pair stored in the key database.

5 SSL certificate expired

Cause:	When establishing an SSL connection with the directory, either the server or client certificate has expired.
Action:	Replace the expired certificate.

6 Unable to initialize SSL

Cause:	Cannot initialize the Secure Sockets Layer
Action:	Check the IBM Global Security Kit (GSKit) is installed and configured, and that the SSL keyfile, password and name are specified correctly.

7 No GSS security context

Cause:	The Generic Security Service (GSSAPI) security context cannot be initialized.
Action:	If the directory server requires authentication using Kerberos, ensure that you have authenticated to the security server. Otherwise, select a different authentication type.

8 Unsupported character set

Cause:	The character set specified to the amtldup program is not supported, or is not valid.
Action:	Use the AMI Administration Tool to update the LDAP directory, or use the default character set for your system.

9 SSL not available

Cause:	The Secure Sockets Layer library cannot be loaded.
Action:	Check the IBM Global Security Kit (GSKit) is installed and configured.

10 SSL handshake failure

Cause:	The AMI cannot connect to the Secure Sockets Layer server.
Action:	Ensure that the directory server is available, then retry the operation.

11 Invalid LDAP URL

Cause:	The LDAP URL, specifying the LDAP server and base to use, is not valid.
Action:	Check the setting of the AMT_REPOSITORY environment variable. To retrieve AMI information from an LDAP directory, this environment variable must specify a valid LDAP URL, as described in the Application Messaging Interface manual.

12 LDAP operations or protocol failure

Cause:	An operational failure occurred while the AMI was performing an LDAP operation.
Action:	This error might be caused by a network failure. Retry the operation when you are sure the server is available on the network.

13 Character conversion failure

Cause:	This error should not occur.
Action:	If the error reoccurs, contact your IBM service representative.

14 Invalid credentials

Cause:	The AMI cannot authenticate to the directory server.
Action:	Ensure that the Bind DN and corresponding password are specified correctly.

15 Missing or invalid bind DN

Cause:	The user Id to use as the Bind DN is not specified, or is not valid.
Action:	Ensure that a valid Bind DN is provided. Note that the full DN (distinguished name) of the directory entry for the user must be specified.

16 Directory server is not available

Cause:	The AMI cannot connect to the LDAP directory.
Action:	Ensure the directory server is running, then retry the operation.

LDAP error codes

17 Unable to initialize connection to directory

Cause: The AMI cannot initialize a connection to the LDAP directory.

Action: Check that the LDAP Server Name (hostname) and port are specified correctly, then retry the operation.

18 Unable to open connection to directory

Cause: The AMI cannot open a connection to the LDAP directory.

Action: Check that the LDAP Server Name (hostname) and port are specified correctly, then retry the operation.

19 Simple bind to directory failed

Cause: The AMI cannot perform a simple bind to the directory.

Action: Ensure that the Bind DN and corresponding password have been specified correctly.

20 SASL bind to directory failed

Cause: A SASL (Simple Authentication and Security Layer) bind to the directory has failed.

Action: Ensure that the directory server supports the authentication type requested.

21 Error on disconnecting from directory

Cause: An error occurred when the AMI was disconnecting from the directory.

Action: No action is required.

22 Missing or invalid base DN

Cause: The distinguished name under which the AMI information is placed in the directory information tree is missing or invalid.

Action: Correct the Base DN specified. Note that when first creating AMI information in the directory, a container with the specified DN will be created if it is not already present. However, the parent DN for the container must already be present.

23 Unable to access directory information

Cause: The AMI cannot access information that should be available from the directory.

Action: Check that network connection to the directory server is still available, and that the directory server is still running.

24 LDAP URL contains invalid base DN

Cause: The LDAP URL contains a base DN that is not valid.

Action: Correct the base DN specified in the LDAP URL. The base DN must specify the DN of a directory object under which AMI information can be found.

25 LDAP URL contains invalid scope

Cause: The LDAP URL contains a scope that is not valid.

Action: Correct or omit the scope specified in the LDAP URL. The recommended scope for AMI searches is sub, that is, search of the whole subtree under the base DN. This is also the AMI default.

26 Insufficient authority

Cause: The user or user Id under which the AMI is accessing the directory (this is normally specified using the bind DN) does not have authority to access or update a required part of the directory.

Action: Either specify a bind DN that has sufficient rights to access to create items in the directory, or ensure that the user Id in use has sufficient access rights to the specified base DN and entries beneath it. For retrieving information, read access is required. For updating AMI information, create and write access is also required.

99 Unexpected LDAP error

Cause: An unexpected error has occurred while accessing the LDAP directory.

Action: If the error reoccurs, contact your IBM service representative.

100 Unable to initialize XML parser

Cause: The Extensible Markup Language (XML) parser cannot be initialized.

Action: This error should not occur. If the error reoccurs, contact your IBM service representative.

101 Error parsing repository file

Cause: When updating AMI information in the directory, an error was detected in the XML repository file, or an AMI dtd file (amt.dtd) is not present in the same directory as the repository xml file.

Action: Use the AMI Administration Tool to create, modify, and update the directory, or ensure that amt.dtd is present in the appropriate directory.

110 Unable to initialize common services

Cause: An internal AMI error has occurred.

Action: If the error reoccurs, contact your IBM service representative.

111 No AMI information changed

Cause: An attempt to update AMI information in the directory was made, but no entries were changed.

Action: No action is required.

LDAP error codes

Appendix B. Constants and structures

This appendix lists the values of the named constants used by the functions described in this manual. For information about WebSphere MQ constants not in this list, see the *WebSphere MQ Application Programming Reference*.

The constants and structures

The constants are grouped according to the parameter or field to which they relate. Names of the constants in a group begin with a common prefix of the form AMxxxx_, where xxxx represents a string of 0 through 4 characters that indicates the nature of the values defined in that group. Within each group, constants are listed in numeric (or alphabetic) order.

Character strings are shown delimited by double quotation marks; the quotation marks are not part of the value.

AMB (Boolean constants)

AMB_FALSE	0L
AMB_TRUE	1L

AMBRW (Browse constants)

AMBRW_UNLOCK	1L
AMBRW_LOCK	2L
AMBRW_FIRST	4L
AMBRW_NEXT	8L
AMBRW_CURRENT	16L
AMBRW_RECEIVE_CURRENT	32L
AMBRW_DEFAULT	AMBRW_NEXT
AMBRW_LOCK_NEXT	(AMBRW_LOCK + AMBRW_NEXT)
AMBRW_LOCK_FIRST	(AMBRW_LOCK + AMBRW_FIRST)
AMBRW_LOCK_CURRENT	(AMBRW_LOCK + AMBRW_CURRENT)

AMCC (Completion codes)

AMCC_OK	0L
AMCC_WARNING	1L
AMCC_FAILED	2L

AMDEF (Service and policy definitions)

AMDEF_POL	"AMT.SYSTEM.POLICY"
AMDEF_PUB	"AMT.SYSTEM.PUBLISHER"
AMDEF_RCV	"AMT.SYSTEM.RECEIVER"
AMDEF_RSP_SND	"AMT.SYSTEM.RESPONSE.SENDER"
AMDEF_SND	"AMT.SYSTEM.SENDER"
AMDEF_SUB	"AMT.SYSTEM.SUBSCRIBER"
AMDEF_SYNC_POINT_POL	"AMT.SYSTEM.SYNCPOINT.POLICY"

AMDT (Definition type constants)

AMDT_UNDEFINED	0L
AMDT_RESPONSE	1L
AMDT_TEMP_DYNAMIC	2L
AMDT_DYNAMIC	3L
AMDT_PREDEFINED	4L

AMENC (Encoding constants)

```
AMENC_NORMAL                     0L
AMENC_REVERSED                   1L
AMENC_NORMAL_FLOAT_390           2L
AMENC_REVERSED_FLOAT_390         3L
AMENC_UNDEFINED                  4L
AMENC_NATIVE          AMENC_NORMAL  (UNIX)
AMENC_NATIVE          AMENC_REVERSED  (WIN32)
AMENC_NATIVE          AMENC_NORMAL_FLOAT_390 (z/OS)
```

AMELEM (AMI C element structure)

This is used in C to add and retrieve AMI RFH name/value pair elements to and from a message.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMELEM_STRUC_ID. The constantAMELEM_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMELEM_VERSION_1.
- `groupBuffLen (AMLONG)`

  Reserved, must be zero.
- `groupLen (AMLONG)`

  Reserved, must be zero.
- `pGroup (AMSTR)`

  Reserved, must be NULL.
- `nameBuffLen (AMLONG)`

  Name buffer length specifies length of the buffer in which the name is to be returned.
- `nameLen (AMLONG)`

  Name length. AMLEN_NULL_TERM specifies that the string is null-terminated.
- `pName (AMSTR)`

  Name.
- `valueBuffLen (AMLONG)`

  Value buffer length.
- `valueLen (AMLONG)`

  Value length. AMLEN_NULL_TERM specifies that the string is null-terminated.
- `pPalue (AMSTR)`

  Value.
- `typeBuffLen (AMLONG)`

  Reserved, must be zero.
- `typeLen (AMLONG)`

  Reserved, must be zero.
- `pType (AMSTR)`

  Reserved, must be NULL.

AMFB (Feedback codes)

```
AMFB_NONE                          0L
AMFB_CODE_EXPIRATION               258L
AMFB_CODE_COA                      259L
AMFB_CODE_COD                      260L
```

AMFMT (Format constants)

```
AMFMT_NONE                       "        "
AMFMT_RF_HEADER                  "MQHRF   "
AMFMT_STRING                     "MQSTR   "
AMFMT_RF2_HEADER                 "MQHRF2  "
```

AMGF and AMGRP (Group status constants)

```
AMGF_IN_GROUP                     1L
AMGF_FIRST                        2L
AMGF_LAST                         4L

AMGRP_MSG_NOT_IN_GROUP            0L
AMGRP_FIRST_MSG_IN_GROUP  ( AMGF_IN_GROUP | AMGF_FIRST )
AMGRP_MIDDLE_MSG_IN_GROUP   AMGF_IN_GROUP
AMGRP_LAST_MSG_IN_GROUP   ( AMGF_IN_GROUP | AMGF_LAST  )
AMGRP_ONLY_MSG_IN_GROUP   ( AMGF_IN_GROUP | AMGF_FIRST | AMGF_LAST )
```

AMH (Handle constants)

```
AMH_NULL_HANDLE       (AMHANDLE) 0L
AMH_INVALID_HANDLE    (AMHANDLE)-1L
```

AMLEN (String length constants)

```
AMLEN_NULL_TERM                  -1L
AMLEN_MAX_NAME_LENGTH           256L
```

AMMCD (Message Content Descriptor tag names)

```
AMMCD_MSG_SERVICE_DOMAIN          "mcd.Msd"
AMMCD_MSG_SET                     "mcd.Set"
AMMCD_MSG_TYPE                    "mcd.Type"
AMMCD_MSG_FORMAT                  "mcd.Fmt"
```

AMMT (Message types)

```
AMMT_NONE                          0L
AMMT_REQUEST                       1L
AMMT_REPLY                         2L
AMMT_REPORT                        4L
AMMT_DATAGRAM                      8L
```

AMPOINTER (Pointer definition)

```
AMPOINTER *
```

AMPS (Publish/subscribe)

Publish/Subscribe constants

Publish/subscribe tag names

```
AMPS_COMMAND                     "MQPSCommand"
AMPS_COMP_CODE                   "MQPSCompCode"
AMPS_DELETE_OPTIONS              "MQPSDelOpts"
AMPS_ERROR_ID                    "MQPSErrorId"
AMPS_ERROR_POS                   "MQPSErrorPos"
```

AMPS_PARAMETER_ID	"MQPSParmId"
AMPS_PUBLICATION_OPTIONS	"MQPSPubOpts"
AMPS_TIMESTAMP	"MQPSPubTime"
AMPS_Q_MGR_NAME	"MQPSQMgrName"
AMPS_Q_NAME	"MQPSQName"
AMPS_REASON	"MQPSReason"
AMPS_REASON_TEXT	"MQPSReasonText"
AMPS_REGISTRATION_OPTIONS	"MQPSRegOpts"
AMPS_SEQUENCE_NUMBER	"MQPSSeqNum"
AMPS_STREAM_NAME	"MQPSStreamName"
AMPS_STRING_DATA	"MQPSStringData"
AMPS_TOPIC	"MQPSTopic"
AMPS_USER_ID	"MQPSUserId"
AMPS_FILTER	"MQPSFilter"
AMPS_SUBSCRIPTION_POINT	"MQPSSubPoint"
AMPS_SEQUENCE	"MQPSSequence"
AMPS_CONTROL	"MQPSControl"

Publish/subscribe tag values

AMPS_ANONYMOUS	"Anon"
AMPS_CORREL_ID_AS_ID	"CorrelAsId"
AMPS_DEREGISTER_ALL	"DeregAll"
AMPS_DIRECT_REQUESTS	"DirectReq"
AMPS_INCLUDE_STREAM_NAME	"InclStreamName"
AMPS_INFORM_IF_RETAINED	"InformIfRet"
AMPS_LOCAL	"Local"
AMPS_NEW_PUBS_ONLY	"NewPubsOnly"
AMPS_PUB_ON_REQUEST_ONLY	"PubOnReqOnly"
AMPS_DELETE_PUBLICATION	"DeletePub"
AMPS_DEREGISTER_PUBLISHER	"DeregPub"
AMPS_DEREGISTER_SUBSCRIBER	"DeregSub"
AMPS_PUBLISH	"Publish"
AMPS_REGISTER_PUBLISHER	"RegPub"
AMPS_REGISTER_SUBSCRIBER	"RegSub"
AMPS_REQUEST_UPDATE	"ReqUpdate"
AMPS_IS_RETAINED_PUBLICATION	"IsRetainedPub"
AMPS_NO_REGISTRATION	"NoReg"
AMPS_NONE	"None"
AMPS_OTHER_SUBSCRIBERS_ONLY	"OtherSubsOnly"
AMPS_RETAIN_PUBLICATION	"RetainPub"
AMPS_PERSISTENT	"Pers"
AMPS_NON_PERSISTENT	"NonPers"
AMPS_PERSISTENT_AS_PUBLISHER	"PersAsPub"
AMPS_PERSISTENT_AS_QUEUE	"PersAsQueue"
AMPS_CC_OK	"0"
AMPS_CC_WARNING	"1"
AMPS_CC_ERROR	"2"

Other publish/subscribe constants

AMPS_APPL_TYPE	"OPT_APP_GRP "
AMPS_MSG_TYPE	"OPT_MSG_TYPE "

AMRC (Reason codes)

AMRC_NONE	0
AMRC_UNEXPECTED_ERR	1
AMRC_INVALID_Q_NAME	2
AMRC_INVALID_SENDER_NAME	3
AMRC_INVALID_RECEIVER_NAME	4
AMRC_INVALID_PUBLISHER_NAME	5
AMRC_INVALID_SUBSCRIBER_NAME	6
AMRC_INVALID_POLICY_NAME	7
AMRC_INVALID_MSG_NAME	8
AMRC_INVALID_SESSION_NAME	9

AMRC_INVALID_DIST_LIST_NAME	10
AMRC_POLICY_HANDLE_ERR	11
AMRC_SERVICE_HANDLE_ERR	12
AMRC_MSG_HANDLE_ERR	13
AMRC_SESSION_HANDLE_ERR	14
AMRC_BROWSE_OPTIONS_ERR	15
AMRC_INSUFFICIENT_MEMORY	16
AMRC_WAIT_TIME_READ_ONLY	17
AMRC_SERVICE_NOT_FOUND	18
AMRC_MSG_NOT_FOUND	19
AMRC_POLICY_NOT_FOUND	20
AMRC_SENDER_NOT_UNIQUE	21
AMRC_RECEIVER_NOT_UNIQUE	22
AMRC_PUBLISHER_NOT_UNIQUE	23
AMRC_SUBSCRIBER_NOT_UNIQUE	24
AMRC_MSG_NOT_UNIQUE	25
AMRC_POLICY_NOT_UNIQUE	26
AMRC_DIST_LIST_NOT_UNIQUE	27
AMRC_RECEIVE_BUFF_PTR_ERR	28
AMRC_RECEIVE_BUFF_LEN_ERR	29
AMRC_SEND_DATA_PTR_ERR	30
AMRC_SEND_DATA_LEN_ERR	31
AMRC_INVALID_IF_SERVICE_OPEN	32
AMRC_SERVICE_ALREADY_OPEN	33
AMRC_DATA_SOURCE_NOT_UNIQUE	34
AMRC_NO_MSG_AVAILABLE	35
AMRC_SESSION_ALREADY_OPEN	36
AMRC_SESSION_ALREADY_CLOSED	37
AMRC_ELEM_NOT_FOUND	38
AMRC_ELEM_COUNT_PTR_ERR	39
AMRC_ELEM_NAME_PTR_ERR	40
AMRC_ELEM_NAME_LEN_ERR	41
AMRC_ELEM_INDEX_ERR	42
AMRC_ELEM_PTR_ERR	43
AMRC_ELEM_STRUC_ERR	44
AMRC_ELEM_STRUC_NAME_ERR	45
AMRC_ELEM_STRUC_VALUE_ERR	46
AMRC_ELEM_STRUC_NAME_BUFF_ERR	47
AMRC_ELEM_STRUC_VALUE_BUFF_ERR	48
AMRC_TRANSPORT_ERR	49
AMRC_TRANSPORT_WARNING	50
AMRC_ENCODING_INCOMPLETE	51
AMRC_ENCODING_MIXED	52
AMRC_ENCODING_ERR	53
AMRC_BEGIN_INVALID	54
AMRC_NO_REPLY_TO_INFO	55
AMRC_SERVICE_ALREADY_CLOSED	56
AMRC_SESSION_NOT_OPEN	57
AMRC_DIST_LIST_INDEX_ERR	58
AMRC_WAIT_TIME_ERR	59
AMRC_SERVICE_NOT_OPEN	60
AMRC_HEADER_TRUNCATED	61
AMRC_HEADER_INVALID	62
AMRC_DATA_LEN_ERR	63
AMRC_BACKOUT_REQUEUE_ERR	64
AMRC_BACKOUT_LIMIT_ERR	65
AMRC_COMMAND_ALREADY_EXISTS	66
AMRC_UNEXPECTED_RECEIVE_ERR	67
AMRC_UNEXPECTED_SEND_ERR	68
AMRC_SENDER_USAGE_ERR	70
AMRC_MSG_TRUNCATED	71
AMRC_CLOSE_SESSION_ERR	72
AMRC_READ_OFFSET_ERR	73
AMRC_RFH_ALREADY_EXISTS	74
AMRC_GROUP_STATUS_ERR	75

AMRC_MSG_ID_LEN_ERR	76
AMRC_MSG_ID_PTR_ERR	77
AMRC_MSG_ID_BUFF_LEN_ERR	78
AMRC_MSG_ID_BUFF_PTR_ERR	79
AMRC_MSG_ID_LEN_PTR_ERR	80
AMRC_CORREL_ID_LEN_ERR	81
AMRC_CORREL_ID_PTR_ERR	82
AMRC_CORREL_ID_BUFF_LEN_ERR	83
AMRC_CORREL_ID_BUFF_PTR_ERR	84
AMRC_CORREL_ID_LEN_PTR_ERR	85
AMRC_FORMAT_LEN_ERR	86
AMRC_FORMAT_PTR_ERR	87
AMRC_FORMAT_BUFF_PTR_ERR	88
AMRC_FORMAT_LEN_PTR_ERR	89
AMRC_FORMAT_BUFF_LEN_ERR	90
AMRC_NAME_BUFF_PTR_ERR	91
AMRC_NAME_LEN_PTR_ERR	92
AMRC_NAME_BUFF_LEN_ERR	93
AMRC_Q_NAME_LEN_ERR	94
AMRC_Q_NAME_PTR_ERR	95
AMRC_Q_NAME_BUFF_PTR_ERR	96
AMRC_Q_NAME_LEN_PTR_ERR	97
AMRC_Q_NAME_BUFF_LEN_ERR	98
AMRC_WAIT_TIME_PTR_ERR	99
AMRC_CCSID_PTR_ERR	100
AMRC_ENCODING_PTR_ERR	101
AMRC_DEFN_TYPE_PTR_ERR	102
AMRC_CCSID_ERR	103
AMRC_DATA_LEN_PTR_ERR	104
AMRC_GROUP_STATUS_PTR_ERR	105
AMRC_DATA_OFFSET_PTR_ERR	106
AMRC_RESP_SENDER_HANDLE_ERR	107
AMRC_RESP_RECEIVER_HANDLE_ERR	108
AMRC_NOT_AUTHORIZED	109
AMRC_TRANSPORT_NOT_AVAILABLE	110
AMRC_BACKED_OUT	111
AMRC_INCOMPLETE_GROUP	112
AMRC_SEND_DISABLED	113
AMRC_SERVICE_FULL	114
AMRC_NOT_CONVERTED	115
AMRC_RECEIVE_DISABLED	116
AMRC_GROUP_BACKOUT_LIMIT_ERR	117
AMRC_SENDER_COUNT_PTR_ERR	118
AMRC_MULTIPLE_REASONS	119
AMRC_NO_RESP_SERVICE	120
AMRC_DATA_PTR_ERR	121
AMRC_DATA_BUFF_LEN_ERR	122
AMRC_DATA_BUFF_PTR_ERR	123
AMRC_DEFN_TYPE_ERR	124
AMRC_BACKOUT_INVALID	125
AMRC_COMMIT_INVALID	126
AMRC_DATA_OFFSET_ERR	127
AMRC_FILE_SYSTEM_ERR	128
AMRC_FILE_ALREADY_EXISTS	129
AMRC_REPORT_CODE_PTR_ERR	130
AMRC_MSG_TYPE_PTR_ERR	131
AMRC_FILE_FORMAT_CONVERTED	132
AMRC_FILE_TRUNCATED	133
AMRC_FILE_NOT_FOUND	134
AMRC_NOT_A_FILE	135
AMRC_FILE_NAME_LEN_ERR	136

AMRC_FILE_NAME_PTR_ERR	137
AMRC_RFH2_FORMAT_ERR	138
AMRC_CCSID_NOT_SUPPORTED	139
AMRC_FILE_MSG_FORMAT_ERR	140
AMRC_MSG_TYPE_NOT_REPORT	141
AMRC_ELEM_STRUC_TYPE_ERR	142
AMRC_ELEM_STRUC_TYPE_BUFF_ERR	143
AMRC_FILE_TRANSFER_INVALID	144
AMRC_FILE_NOT_WRITTEN	145
AMRC_FILE_FORMAT_NOT_SUPPORTED	146
AMRC_NEGATIVE_RECEIVE_BUFF_LEN	147
AMRC_LIBRARY_NOT_FOUND	148
AMRC_LIBRARY_FUNCTION_PTR_ERR	149
AMRC_LIBRARY_INV_POINT_ERR	150
AMRC_LIBRARY_DUP_FUNCTION	151
AMRC_POLICY_HANDLER_ERR	152
AMRC_POLICY_HANDLER_WARNING	153
AMRC_REPORT_CODE_ERR	154
AMRC_GROUP_MSG_RETURNED	155
AMRC_NON_GROUP_MSG_IN_GROUP	156
AMRC_INVALID_TRACE_LEVEL	400
AMRC_CONN_NAME_NOT_FOUND	401
AMRC_HOST_FILE_NOT_FOUND	402
AMRC_HOST_FILENAME_ERR	403
AMRC_HOST_FILE_ERR	404
AMRC_POLICY_NOT_IN_REPOS	405
AMRC_SENDER_NOT_IN_REPOS	406
AMRC_RECEIVER_NOT_IN_REPOS	407
AMRC_DIST_LIST_NOT_IN_REPOS	408
AMRC_PUBLISHER_NOT_IN_REPOS	409
AMRC_SUBSCRIBER_NOT_IN_REPOS	410
AMRC_RESERVED_NAME_IN_REPOS	411
AMRC_REPOS_FILENAME_ERR	414
AMRC_REPOS_WARNING	415
AMRC_REPOS_ERR	416
AMRC_REPOS_NOT_FOUND	418
AMRC_TRANSPORT_LIBRARY_ERR	419
AMRC_HOST_CACHE_ERR	420
AMRC_REPOS_CACHE_ERR	421
AMRC_PRIMARY_HANDLE_ERR	422
AMRC_SESSION_EXPIRED	423
AMRC_DTD_NOT_FOUND	424
AMRC_LDAP_ERR	425

The following AMRC values are returned exclusively by extended C AMI functions.

AMRC_ACCEPT_DIRECT_ERR	201
AMRC_ACCEPT_DIRECT_PTR_ERR	202
AMRC_ACCEPT_TRUNCATED_ERR	203
AMRC_ACCEPT_TRUNCATED_PTR_ERR	204
AMRC_ANON_ERR	205
AMRC_ANON_PTR_ERR	206
AMRC_APPL_GROUP_BUFF_LEN_ERR	207
AMRC_APPL_GROUP_BUFF_PTR_ERR	208
AMRC_APPL_GROUP_LEN_ERR	209
AMRC_APPL_GROUP_LEN_PTR_ERR	210
AMRC_APPL_GROUP_PTR_ERR	211
AMRC_BIND_ON_OPEN_ERR	212
AMRC_BIND_ON_OPEN_PTR_ERR	213

AMRC_CHL_NAME_BUFF_LEN_ERR	214
AMRC_CHL_NAME_BUFF_PTR_ERR	215
AMRC_CHL_NAME_LEN_ERR	216
AMRC_CHL_NAME_LEN_PTR_ERR	217
AMRC_CHL_NAME_PTR_ERR	218
AMRC_CLOSE_DELETE_ERR	219
AMRC_CLOSE_DELETE_PTR_ERR	220
AMRC_CONTEXT_ERR	221
AMRC_CONTEXT_PTR_ERR	222
AMRC_CONVERT_ERR	223
AMRC_CONVERT_PTR_ERR	224
AMRC_COUNT_ERR	225
AMRC_COUNT_PTR_ERR	226
AMRC_CUST_PARM_BUFF_LEN_ERR	227
AMRC_CUST_PARM_BUFF_PTR_ERR	228
AMRC_CUST_PARM_LEN_ERR	229
AMRC_CUST_PARM_LEN_PTR_ERR	230
AMRC_CUST_PARM_PTR_ERR	231
AMRC_DLY_PERSISTENCE_ERR	232
AMRC_DLY_PERSISTENCE_PTR_ERR	233
AMRC_DST_SUPPORT_ERR	234
AMRC_DST_SUPPORT_PTR_ERR	235
AMRC_EXPIRY_ERR	236
AMRC_EXPIRY_PTR_ERR	237
AMRC_FILE_DISP_ERR	238
AMRC_FILE_DISP_PTR_ERR	239
AMRC_FILE_RCD_LEN_ERR	240
AMRC_FILE_RCD_LEN_PTR_ERR	241
AMRC_GROUP_ID_BUFF_LEN_ERR	242
AMRC_GROUP_ID_BUFF_PTR_ERR	243
AMRC_GROUP_ID_LEN_ERR	244
AMRC_GROUP_ID_LEN_PTR_ERR	245
AMRC_GROUP_ID_PTR_ERR	246
AMRC_HANDLE_POISON_MSG_ERR	247
AMRC_HANDLE_POISON_MSG_PTR_ERR	248
AMRC_HANDLE_PTR_ERR	249
AMRC_IMPL_OPEN_ERR	250
AMRC_IMPL_OPEN_PTR_ERR	251
AMRC_INFORM_IF_RET_ERR	252
AMRC_INFORM_IF_RET_PTR_ERR	253
AMRC_INTERVAL_ERR	254
AMRC_INTERVAL_PTR_ERR	255
AMRC_LEAVE_OPEN_ERR	256
AMRC_LEAVE_OPEN_PTR_ERR	257
AMRC_LOCAL_ERR	258
AMRC_LOCAL_PTR_ERR	259
AMRC_MCD_PARM_BUFF_LEN_ERR	260
AMRC_MCD_PARM_BUFF_PTR_ERR	261
AMRC_MCD_PARM_LEN_ERR	262
AMRC_MCD_PARM_LEN_PTR_ERR	263
AMRC_MCD_PARM_PTR_ERR	264
AMRC_MGR_NAME_BUFF_LEN_ERR	265
AMRC_MGR_NAME_BUFF_PTR_ERR	266
AMRC_MGR_NAME_LEN_ERR	267
AMRC_MGR_NAME_LEN_PTR_ERR	268
AMRC_MGR_NAME_PTR_ERR	269
AMRC_MSG_LEN_ERR	270
AMRC_MSG_LEN_PTR_ERR	271
AMRC_MSG_TYPE_ERR	272

```
AMRC_NEW_CORREL_ID_ERR         273
AMRC_NEW_CORREL_ID_PTR_ERR     274
AMRC_NEW_PUBS_ONLY_ERR         275
AMRC_NEW_PUBS_ONLY_PTR_ERR     276
AMRC_PERSISTENCE_ERR           277
AMRC_PERSISTENCE_PTR_ERR       278
AMRC_PRIORITY_ERR              279

AMRC_PRIORITY_PTR_ERR          280
AMRC_PUB_ON_REQ_ERR            281
AMRC_PUB_ON_REQ_PTR_ERR        282
AMRC_PUB_OTHERS_ONLY_ERR       283
AMRC_PUB_OTHERS_ONLY_PTR_ERR   284
AMRC_READ_ONLY_ERR             285
AMRC_READ_ONLY_PTR_ERR         286
AMRC_REMOVE_ALL_ERR            287
AMRC_REMOVE_ALL_PTR_ERR        288
AMRC_REPORT_OPTION_ERR         289

AMRC_REPORT_OPTION_PTR_ERR     290
AMRC_RETAIN_ERR                291
AMRC_RETAIN_PTR_ERR            292
AMRC_SEGMENT_ERR               293
AMRC_SEGMENT_PTR_ERR           294
AMRC_SEQ_NO_ERR                295
AMRC_SEQ_NO_PTR_ERR            296
AMRC_SET_NAME_INVALID          297
AMRC_SHARED_ERR                298
AMRC_SHARED_PTR_ERR            299

AMRC_SND_TYPE_ERR              300
AMRC_SND_TYPE_PTR_ERR          301
AMRC_SRV_TYPE_ERR              302
AMRC_SRV_TYPE_PTR_ERR          303
AMRC_SPLIT_LOGICAL_ERR         304
AMRC_SPLIT_LOGICAL_PTR_ERR     305
AMRC_SUBS_POINT_BUFF_LEN_ERR   306
AMRC_SUBS_POINT_BUFF_PTR_ERR   307
AMRC_SUBS_POINT_LEN_ERR        308
AMRC_SUBS_POINT_LEN_PTR_ERR    309

AMRC_SUBS_POINT_PTR_ERR        310
AMRC_SUPPRESS_REG_ERR          311
AMRC_SUPPRESS_REG_PTR_ERR      312
AMRC_SYNCPOINT_ERR             313
AMRC_SYNCPOINT_PTR_ERR         314
AMRC_TCP_ADDR_BUFF_LEN_ERR     315
AMRC_TCP_ADDR_BUFF_PTR_ERR     316
AMRC_TCP_ADDR_LEN_ERR          317
AMRC_TCP_ADDR_LEN_PTR_ERR      318
AMRC_TCP_ADDR_PTR_ERR          319

AMRC_TRP_TYPE_ERR              320
AMRC_TRP_TYPE_PTR_ERR          321
AMRC_TRUSTED_ERR               322
AMRC_TRUSTED_PTR_ERR           323
AMRC_USE_CORREL_ID_ERR         324
AMRC_USE_CORREL_ID_PTR_ERR     325
AMRC_WAIT_WHOLE_GROUP_ERR      326
AMRC_WAIT_WHOLE_GROUP_PTR_ERR  327
AMRC_CON_INT_PROP_ID_ERR       328
AMRC_CON_STR_PROP_ID_ERR       329
```

AMRC_MSG_INT_PROP_ID_ERR	330
AMRC_MSG_STR_PROP_ID_ERR	331
AMRC_POLICY_INT_PROP_ID_ERR	332
AMRC_POLICY_STR_PROP_ID_ERR	333
AMRC_SRV_INT_PROP_ID_ERR	334
AMRC_SRV_STR_PROP_ID_ERR	335
AMRC_INVALID_IF_CON_OPEN	336
AMRC_CON_HANDLE_ERR	337

The following AMRC values are applicable only to the Java programming language.

AMRC_JAVA_FIELD_ERR	500
AMRC_JAVA_METHOD_ERR	501
AMRC_JAVA_CLASS_ERR	502
AMRC_JAVA_JNI_ERR	503
AMRC_JAVA_CREATE_ERR	504
AMRC_JAVA_NULL_PARM_ERR	505

AMSD (System default names and handle synonyms)

System default names and handles

Default names

AMSD_POL	"SYSTEM.DEFAULT.POLICY"
AMSD_PUB	"SYSTEM.DEFAULT.PUBLISHER"
AMSD_PUB_SND	"SYSTEM.DEFAULT.PUBLISHER"
AMSD_RCV	"SYSTEM.DEFAULT.RECEIVER"
AMSD_RCV_MSG	"SYSTEM.DEFAULT.RECEIVE.MESSAGE"
AMSD_RSP_SND	"SYSTEM.DEFAULT.RESPONSE.SENDER"
AMSD_SND	"SYSTEM.DEFAULT.SENDER"
AMSD_SND_MSG	"SYSTEM.DEFAULT.SEND.MESSAGE"
AMSD_SESSION_NAME	"SYSTEM.DEFAULT.SESSION"
AMSD_SUB	"SYSTEM.DEFAULT.SUBSCRIBER"
AMSD_SUB_SND	"SYSTEM.DEFAULT.SUBSCRIBER"
AMSD_SUB_RCV	"SYSTEM.DEFAULT.SUBSCRIBER.RECEIVER"
AMSD_SYNC_POINT_POL	"SYSTEM.DEFAULT.SYNCPOINT.POLICY"

Default handle synonyms

AMSD_RSP_SND_HANDLE	(AMHSND)-5L
AMSD_RCV_HANDLE	(AMHRCV)-6L
AMSD_POL_HANDLE	(AMHPOL)-7L
AMSD_SYNC_POINT_POL_HANDLE	(AMHPOL)-8L
AMSD_SND_MSG_HANDLE	(AMHMSG)-9L
AMSD_RCV_MSG_HANDLE	(AMHMSG)-10L

AMWT (Wait time constant)

AMWT_UNLIMITED	-1L

C constants used by extended C AMI functions

This section lists C constants that are defined in amtxc.h and that are used exclusively by extended C AMI functions.

AMCON (Connection object integer property value constants)

AMCON_NO	AMPROP_FALSE
AMCON_YES	AMPROP_TRUE
AMCON_TRP_TYPE_MQ_CLIENT	1L
AMCON_TRP_TYPE_MQ_SERVER	2L
AMCON_TRP_TYPE_MQ_AUTO	3L
AMCON_TRP_TYPE_OTHER	4L

AMCON_INT (Connection object integer property identifiers)

AMCON_INT_TRP_TYPE	101L
AMCON_INT_CCSID	102L
AMCON_INT_DST_SUPPORT	103L
AMCON_INT_TRUSTED	104L
AMCON_INT_MQHOBJ	105L
AMCON_INT_MQHCON	106L
AMCON_INT_USER_CONTEXT	107L
AMCON_INT_PROP_LAST	AMCON_INT_USER_CONTEXT

AMCON_STR (Connection object string property identifiers)

AMCON_STR_NAME	1001L
AMCON_STR_MGR_NAME	1002L
AMCON_STR_REAL_MGR_NAME	1003L
AMCON_STR_CLIENT_CHL_NAME	1004L
AMCON_STR_SERVER_TCP_ADDR	1005L
AMCON_STR_PROP_LAST	AMCON_STR_SERVER_TCP_ADDR

AMEI (Expiry interval constant)

AMEI_UNLIMITED	(-1L)

AMH (Handle constants)

AMH_INVALID_HCON	(-1L)
AMH_INVALID_HOBJ	(-1L)

AMH (Handle property limit constants for all object types)

AMH_MIN	0x80000000
AMH_MAX	0x7FFFFFFF

AMINT (Positive integer property limit constants for all object types)

AMINT_PROP_MIN	0L
AMINT_PROP_MAX	999999999L

AMINV (Invocation points)

AMINV_CONNECTION_OPEN	1L
AMINV_CONNECTION_CLOSE	2L
AMINV_BEGIN	3L
AMINV_ROLLBACK	4L
AMINV_COMMIT	5L
AMINV_SENDER_OPEN	6L
AMINV_SENDER_CLOSE	7L
AMINV_SENDER_SEND	8L
AMINV_RECEIVER_OPEN	9L

```
AMINV_RECEIVER_CLOSE        10L
AMINV_RECEIVER_RECEIVE      11L
AMINV_DIST_LIST_OPEN        12L
AMINV_DIST_LIST_CLOSE       13L
AMINV_DIST_LIST_SEND        14L
AMINV_HANDLE_POISON_MSG     15L

AMINV_TRANSPORT_FIRST       1000L
AMINV_PRE_MQBACK            1001L
AMINV_POST_MQBACK           1002L
AMINV_PRE_MQBEGIN           1003L
AMINV_POST_MQBEGIN          1004L
AMINV_PRE_MQCLOSE           1005L
AMINV_POST_MQCLOSE          1006L
AMINV_PRE_MQCMIT            1007L
AMINV_POST_MQCMIT           1008L
AMINV_PRE_MQCONN            1009L
AMINV_POST_MQCONN           1010L
AMINV_PRE_MQCONNX           1011L
AMINV_POST_MQCONNX          1012L
AMINV_PRE_MQDISC            1013L
AMINV_POST_MQDISC           1014L
AMINV_PRE_MQGET             1015L
AMINV_POST_MQGET            1016L
AMINV_PRE_MQINQ             1017L
AMINV_POST_MQINQ            1018L
AMINV_PRE_MQOPEN            1019L
AMINV_POST_MQOPEN           1020L
AMINV_PRE_MQPUT             1021L
AMINV_POST_MQPUT            1022L
AMINV_PRE_MQPUT1            1023L
AMINV_POST_MQPUT1           1024L
AMINV_PRE_MQSET             1025L
AMINV_POST_MQSET            1026L
```

AMLONG (Signed integer property limit constants for all object types)

```
AMLONG_MIN                 0x80000000
AMLONG_MAX                 0x7FFFFFFF
```

AMMSG (Assemble message options for use with amxMsgAssemble)

```
AMMSG_AMO_NONE              0L
AMMSG_AMO_FORMAT_RFH        1L
AMMSG_AMO_FORMAT_RFH2       2L
AMMSG_AMO_EXT_HDR           4L
```

AMMSG (Message object integer property value constants)

```
AMMSG_NO                    AMPROP_FALSE
AMMSG_YES                   AMPROP_TRUE

AMMSG_CCSID_INHERIT         (-2L)
AMMSG_CCSID_EMBEDDED        (-1L)
AMMSG_CCSID_DFT             0L

AMMSG_SEQ_NO_FIRST          1L

AMMSG_ORIGINAL_LEN_UNDEFINED (-1L)
```

AMMSG_RO_EXCPTN_NO	0L
AMMSG_RO_EXCPTN_YES	1L
AMMSG_RO_EXCPTN_DATA	2L
AMMSG_RO_EXCPTN_FULL	3L

AMMSG_RO_EXPIRY_NO	0L
AMMSG_RO_EXPIRY_YES	1L
AMMSG_RO_EXPIRY_DATA	2L
AMMSG_RO_EXPIRY_FULL	3L

AMMSG_RO_COA_NO	0L
AMMSG_RO_COA_YES	1L
AMMSG_RO_COA_DATA	2L
AMMSG_RO_COA_FULL	3L

AMMSG_RO_COD_NO	0L
AMMSG_RO_COD_YES	1L
AMMSG_RO_COD_DATA	2L
AMMSG_RO_COD_FULL	3L

AMMSG_INT (Message object integer property identifiers)

AMMSG_INT_BACKOUT_COUNT	101L
AMMSG_INT_CCSID	102L
AMMSG_INT_MSG_DATA_LEN	103L
AMMSG_INT_MSG_LEN	104L
AMMSG_INT_ORIGINAL_LEN	105L
AMMSG_INT_DATA_OFFSET	106L
AMMSG_INT_INT_ENCODING	107L
AMMSG_INT_DEC_ENCODING	108L
AMMSG_INT_FLOAT_ENCODING	109L
AMMSG_INT_EXPIRY	110L
AMMSG_INT_FB	111L
AMMSG_INT_MSG_TYPE	112L
AMMSG_INT_SEQ_NO	113L
AMMSG_INT_PERSISTENT	114L
AMMSG_INT_PRIORITY	115L
AMMSG_INT_RO_EXCEPTION	116L
AMMSG_INT_RO_EXPIRY	117L
AMMSG_INT_RO_COA	118L
AMMSG_INT_RO_COD	119L
AMMSG_INT_RO_COPY_MSG_ID	120L
AMMSG_INT_RO_DISCARD	121L
AMMSG_INT_RO_PAN	122L
AMMSG_INT_RO_NAN	123L
AMMSG_INT_RO_NEW_MSG_ID	124L
AMMSG_INT_SEGMENTS	125L
AMMSG_INT_IN_GROUP	126L
AMMSG_INT_APPL_CCSID	127L
AMMSG_INT_USER_CONTEXT	128L
AMMSG_INT_PROP_LAST	AMMSG_INT_USER_CONTEXT

AMMSG_STR (Message object string property identifiers)

AMMSG_STR_NAME	1001L
AMMSG_STR_FORMAT	1002L
AMMSG_STR_CORREL_ID	1003L
AMMSG_STR_MSG_ID	1004L
AMMSG_STR_GROUP_ID	1005L
AMMSG_STR_PROP_LAST	AMMSG_STR_GROUP_ID

AMOP (AMI operation codes)

AMOP_CONNECTION_OPEN	1L
AMOP_CONNECTION_CLOSE	2L
AMOP_BEGIN	3L

AMOP_ROLLBACK	4L
AMOP_COMMIT	5L
AMOP_SENDER_OPEN	6L
AMOP_SENDER_CLOSE	7L
AMOP_SENDER_SEND	8L
AMOP_RECEIVER_OPEN	9L
AMOP_RECEIVER_CLOSE	10L
AMOP_RECEIVER_RECEIVE	11L
AMOP_DIST_LIST_OPEN	12L
AMOP_DIST_LIST_CLOSE	13L
AMOP_DIST_LIST_SEND	14L
AMOP_HANDLE_POISON_MSG	15L

AMPOL (Policy object integer property value constants)

AMPOL_NO	AMPROP_FALSE
AMPOL_YES	AMPROP_TRUE

AMPOL_CONTEXT_NONE	1L
AMPOL_CONTEXT_AS_TRP	2L
AMPOL_CONTEXT_PASS_ID	3L
AMPOL_CONTEXT_PASS_ALL	4L

AMPOL_PRIORITY_AS_TRP	(-1L)

AMPOL_PERSISTENCE_AS_TRP	(-1L)
AMPOL_PERSISTENCE_NO	0L
AMPOL_PERSISTENCE_YES	1L

AMPOL_REPORT_DATA_NO	1L
AMPOL_REPORT_DATA_YES	2L
AMPOL_REPORT_DATA_FULL	3L

AMPOL_BIND_ON_OPEN_AS_TRP	(-1L)
AMPOL_BIND_ON_OPEN_NO	0L
AMPOL_BIND_ON_OPEN_YES	1L

AMPOL_CLOSE_DELETE_NO	0L
AMPOL_CLOSE_DELETE_YES	1L
AMPOL_CLOSE_DELETE_PURGE	2L

AMPOL_FILE_DISP_NEW	0L
AMPOL_FILE_DISP_OVERWRITE	1L
AMPOL_FILE_DISP_APPEND	2L

AMPOL_DLY_PERSISTENCE_AS_PUB	(-2L)
AMPOL_DLY_PERSISTENCE_AS_TRP	(-1L)
AMPOL_DLY_PERSISTENCE_NO	0L
AMPOL_DLY_PERSISTENCE_YES	1L

AMPOL_INT (Policy object integer property identifiers)

AMPOL_INT_SYNCPOINT	101L
AMPOL_INT_CONTEXT	102L
AMPOL_INT_SND_IMPL_OPEN	103L
AMPOL_INT_SND_LEAVE_OPEN	104L
AMPOL_INT_PRIORITY	105L
AMPOL_INT_PERSISTENCE	106L
AMPOL_INT_EXPIRY	107L
AMPOL_INT_RETRY_COUNT	108L
AMPOL_INT_RETRY_INTERVAL	109L
AMPOL_INT_REPORT_DATA	110L
AMPOL_INT_RO_EXCEPTION	111L
AMPOL_INT_RO_EXPIRY	112L
AMPOL_INT_RO_COA	113L
AMPOL_INT_RO_COD	114L

AMPOL_INT_RO_COPY_MSG_ID	115L
AMPOL_INT_RO_DISCARD	116L
AMPOL_INT_NEW_CORREL_ID	117L
AMPOL_INT_SEGMENT	118L
AMPOL_INT_SPLIT_LOGICAL	119L
AMPOL_INT_FILE_RCD_LEN	120L
AMPOL_INT_BIND_ON_OPEN	121L
AMPOL_INT_RCV_IMPL_OPEN	122L
AMPOL_INT_RCV_LEAVE_OPEN	123L
AMPOL_INT_CLOSE_DELETE	124L
AMPOL_INT_WAIT_INTERVAL	125L
AMPOL_INT_WAIT_INTERVAL_RO	126L
AMPOL_INT_CONVERT	127L
AMPOL_INT_WAIT_WHOLE_GROUP	128L
AMPOL_INT_HANDLE_POISON_MSG	129L
AMPOL_INT_ACCEPT_TRUNCATED	130L
AMPOL_INT_SHARED	131L
AMPOL_INT_FILE_DISP	132L
AMPOL_INT_RETAIN_PUBS	133L
AMPOL_INT_PUB_OTHERS_ONLY	134L
AMPOL_INT_SUPPRESS_REG	135L
AMPOL_INT_PUB_LOCAL	136L
AMPOL_INT_ACCEPT_DIRECT	137L
AMPOL_INT_PUB_ANON	138L
AMPOL_INT_PUB_USE_CORREL_ID	139L
AMPOL_INT_SUB_LOCAL	140L
AMPOL_INT_NEW_PUBS_ONLY	141L
AMPOL_INT_PUB_ON_REQ	142L
AMPOL_INT_INFORM_IF_RET	143L
AMPOL_INT_REMOVE_ALL_SUBS	144L
AMPOL_INT_SUB_ANON	145L
AMPOL_INT_SUB_USE_CORREL_ID	146L
AMPOL_INT_DLY_PERSISTENCE	147L
AMPOL_INT_USER_CONTEXT	148L
AMPOL_INT_PROP_LAST	AMPOL_INT_USER_CONTEXT

AMPOL_STR (Policy object string property identifiers)

AMPOL_STR_NAME	1001L
AMPOL_STR_APPL_GROUP	1002L
AMPOL_STR_SUBS_POINT	1003L
AMPOL_STR_PROP_LAST	AMPOL_STR_SUBS_POINT

AMPROP (Integer property true/false constants)

AMPROP_FALSE	0L
AMPROP_TRUE	1L

AMSRV (Message object integer property value constants)

AMSRV_NO	AMPROP_FALSE
AMSRV_YES	AMPROP_TRUE
AMSRV_TYPE_RCV	1L
AMSRV_TYPE_SND	2L
AMSRV_TYPE_DST_SND	3L
AMSRV_TYPE_PUB_SND	4L
AMSRV_TYPE_SUB_SND	5L
AMSRV_TYPE_SUB_RCV	6L
AMSRV_SND_TYPE_NATIVE	1L
AMSRV_SND_TYPE_MQINT_V1	2L
AMSRV_SND_TYPE_MQINT_V2	3L
AMSRV_SND_TYPE_RFH1	4L
AMSRV_SND_TYPE_RFH2	5L
AMSRV_CCSID_UNDEFINED	0L

AMSRV_INT (Service object integer property identifiers)

AMSRV_INT_TYPE	101L
AMSRV_INT_SND_TYPE	102L
AMSRV_INT_DEFN_TYPE	103L
AMSRV_INT_ENCODING	104L
AMSRV_INT_CCSID	105L
AMSRV_INT_NEXT_SND_HANDLE	106L
AMSRV_INT_SUB_RCV_HANDLE	107L
AMSRV_INT_MQHOBJ	108L
AMSRV_INT_USER_CONTEXT	109L
AMSRV_INT_PROP_LAST	AMSRV_INT_USER_CONTEXT

AMSRV_STR (Service object string property identifiers)

AMSRV_STR_NAME	1001L
AMSRV_STR_Q_NAME	1002L
AMSRV_STR_MGR_NAME	1003L
AMSRV_STR_MODEL_Q_NAME	1004L
AMSRV_STR_DFT_MSG_FORMAT	1005L
AMSRV_STR_DYNAMIC_PREFIX	1006L
AMSRV_STR_DFT_MCD_DOMAIN	1007L
AMSRV_STR_DFT_MCD_SET	1008L
AMSRV_STR_DFT_MCD_TYPE	1009L
AMSRV_STR_DFT_MCD_FORMAT	1010L
AMSRV_STR_CUST_PARMS	1011L
AMSTR_STR_PROP_LAST	AMSRV_STR_CUST_PARMS

AMSTR (Maximum string length constant for all object types)

AMSTR_MAX_LEN	512L

AMTC (Trace control constants)

AMTC_TEXT	0L
AMTC_FUNCTION_ENTRY	1L
AMTC_FUNCTION_EXIT	2L
AMTC_DEFAULT	AMTC_TEXT

C constants and AMI parameter structures used by policy handlers

This section lists C constants that are defined in amtphlc.h and that are used exclusively by policy handlers.

AMPH (Policy handler continuation codes)

```
AMPH_CONTINUE               0L
AMPH_COMPLETE               1L
```

AMPH (Policy handler transport types)

```
AMPH_TRANSPORT_TYPE_MQ       "MQSeries"
AMPH_TRANSPORT_LEN_MQ        9
AMPH_TRANSPORT_TYPE_DEFAULT  AMPH_TRANSPORT_TYPE_MQ
```

AMPH (Policy handler maximum lengths)

```
AMPH_MAX_TRANSPORT_LENGTH    256L
```

AMRO (Receive options)

```
AMRO_ BROWSE        1L
AMRO_RECEIVE        2L
AMRO_RECEIVE_FILE   3L
```

AMPHBGN (AMI C begin parameter structure)

This is used with AMIP_BEGIN.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHBGN_STRUC_ID The constant AMPHBGN_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHBGN_VERSION_1.
- `hCon (AMHCON)`

  The handle of the AMI connection object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.

AMPHCLC (AMI C close connection parameter structure)

This is used with AMIP_CONNECTION_CLOSED.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHCLC_STRUC_ID. The constant AMPHCLC_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHBGN_VERSION_1.
- `hCon (AMHCON)`

  The handle of the AMI connection object.
- `hPolicy (AMHPOL)`

The handle of the AMI policy object.

AMPHCLD (AMI C close distribution list parameter structure)

This is used with AMIP_DIST_LIST_CLOSE.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHCLD_STRUC_ID. The constant AMPHCLD_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHCLD_VERSION_1.
- `hDstList (AMHDST)`

  The handle of the AMI distribution list object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.

AMPHCLS (AMI C close service parameter structure)

This is used with AMIP_SENDER_CLOSE or AMIP_RECEIVER_CLOSE.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHCLS_STRUC_ID. The constant AMPHCLS_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHCLS_VERSION_1.
- `hService (AMHSRV)`

  The handle of the AMI (sender or receiver) service object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.

AMPHCMT (AMI C commit parameter structure)

This is used with AMIP_COMMIT.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHCMT_STRUC_ID. The constant AMPHCMT_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHCMT_VERSION_1.
- `hCon (AMHCON)`

  The handle of the AMI connection object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.

AMPHHPM (AMI C handle poison message parameter structure)

This is used with AMIP_HANDLE_POISON_MESSAGE.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHHPM_STRUC_ID. The constant AMPHHPM_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHHPM_VERSION_1.
- `hService (AMHSRV)`

  The handle of the AMI (receiver) service object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.
- `pData (PAMBYTE)`

  Pointer message data.
- `dataLen (AMLONG)`

  The length of the message data in bytes.

AMPHOPC (AMI C open connection parameter structure)

This is used with AMIP_CONNECTION_OPEN.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHOPC_STRUC_ID. The constant AMPHOPC_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHOPC_VERSION_1.
- `hCon (AMHCON)`

  The handle of the AMI connection object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.

AMPHOPD (AMI C open distribution list parameter structure)

This is used with AMIP_DIST_LIST_OPEN.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHOPD_STRUC_ID. The constant AMPHOPD_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHOPD_VERSION_1.
- `hDstList (AMHDST)`

  The handle of the AMI distribution list service object.

- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.

AMPHOPS (AMI C open service parameter structure)

This is used with AMIP_SENDER_OPEN and AMIP_RECEIVER_OPEN.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHOPS_STRUC_ID. The constant AMPHOPS_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHOPS_VERSION_1.
- `hService (AMHSRV)`

  The handle of the AMI (sender or receiver) service object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.

This is used with

Fields

- `strucId (AMCHAR8)`

  Structure identifier.
- `version (AMLONG)`

  Structure version number.
- `hCon (AMHCON)`

  The handle of the AMI connection object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.

AMPHPARM (AMI parameter union)

This is a union of the following parameter structures:

- `bgnParms (AMPHBGN)`

  The parameter structure for Begin.
- `clcParms (AMPHCLC)`

  The parameter structure for Close Connection.
- `cldParms (AMPHCLD)`

  The parameter structure for Close Distribution List.
- `clsParms (AMPHCLS)`

  The parameter structure for Close Service.
- `cmtParms (AMPHCMT)`

  The parameter structure for Commit.
- `hpmParms (AMPHHPM)`

  The parameter structure for Handle Poison Message.
- `opcParms (AMPHOPC)`

  The parameter structure for Open Connection.
- `opdParms (AMPHOPD)`

  The parameter structure for Open Distribution List.

- `opsParms (AMPHOPS)`

  The parameter structure for Open Service.
- `rbkParms (AMPHRBK)`

  The parameter structure for Rollback.
- `rcsParms (AMPHRCS)`

  The parameter structure for Receiver from Service.
- `sndParms (AMPHSND)`

  The parameter structure for Send To Distribution List.
- `snsParms (AMPHSNS)`

  The parameter structure for Send to Service.

AMPHRBK (AMI C rollback parameter structure)

This is used with AMIP_ROLLBACK.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHRBK_STRUC_ID. The constant AMPHRBK_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHRBK_VERSION_1.
- `hCon (AMHCON)`

  The handle of the AMI connection object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.

AMPHRCS (AMI C receive from service parameter structure)

This is used with AMIP_RECEIVER_RECEIVE.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHRCS_STRUC_ID. The constant AMPHRCS_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHRCS_VERSION_1.
- `hService (AMHSRV)`

  The handle of the AMI (receiver) service object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.
- `rcvOpts (AMLONG)`

  Receive options (AMRO_BROWSE or AMRO_RECEIVE).
- `brwOpts (AMHPOL)`

  Browse options.
- `hSelMsg (AMHMSG)`

  The handle of the selection message.

- `buffLen (AMLONG)`

  The length of the message data buffer in bytes.
- `pDataLen (PAMLONG)`

  Pointer to length of message data.
- `hMsg (AMHMSG)`

  The handle of the message object.
- `hRespService (AMHSRV)`

  Response (sender) service handle.

AMPHSND (AMI C send to distribution list parameter structure)

This is used with AMIP_DIST_LIST_SEND.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHSND_STRUC_ID. The constant AMPHSND_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHSND_VERSION_1.
- `hDstList (AMHDST)`

  The handle of the AMI (sender or receiver) service object.
- `hResponse (AMHSRV)`

  The handle of the response receiver service object.
- `dataLen (AMLONG)`

  The length of the message data in bytes.
- `pData (PAMBYTE)`

  Pointer to message data.
- `hMsg (AMHMSG)`

  The handle of the message object.

AMPHSNS (AMI C send to service parameter structure)

This is used with AMIP_SENDER_SEND.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHSNS_STRUC_ID. The constant AMPHSNS_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHSNS_VERSION_1.
- `hService (AMHSRV)`

  The handle of the AMI (sender) service object.
- `hPolicy (AMHPOL)`

  The handle of the AMI policy object.
- `hResponse (AMHSRV)`

  The handle of the response receiver service object.

- `hReqMsg (AMHMSG)`
  The handle of the request message.
- `dataLen (AMLONG)`
  The length of the message data in bytes.
- `pData (PAMBYTE)`
  Pointer to message data.
- `hMsg (AMHMSG)`
  The handle of the message object.

C constants and MQI parameter structures used by policy handlers

AMPHMQBACK (AMI C MQBACK parameter structure)

This is used with AMIP_PRE_MQBACK and AMIP_POST_MQBACK.

Fields

- `strucId (AMCHAR8)`
  Structure identifier. The value must be AMPHMQBACK_STRUC_ID. The constant AMPHMQBACK_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`
  Structure version number. The value must be AMPHMQBACK_VERSION_1.
- `pHconn (PMQHCONN)`
  Pointer to the WebSphere MQ connection handle.
- `pCompCode (PMQLONG)`
  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`
  Pointer to the WebSphere MQ reason code.

AMPHMQBEGIN (AMI C MQBEGIN parameter structure)

This is used with AMIP_PRE_MQBEGIN and AMIP_POST_MQBEGIN.

Fields

- `strucId (AMCHAR8)`
  Structure identifier. The value must be AMPHMQBEGIN_STRUC_ID. The constant AMPHMQBEGIN_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`
  Structure version number. The value must be AMPHMQBACK_VERSION_1.
- `pHconn (PMQHCONN)`
  Pointer to the WebSphere MQ connection handle.
- `pBeginOptions (PMQVOID)`
  Pointer to the MQBEGIN options.
- `pCompCode (PMQLONG)`
  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`
  Pointer to the WebSphere MQ reason code.

AMPHMQCLOSE (AMI C MQCLOSE parameter structure)

This is used with AMIP_PRE_MQCLOSE and AMIP_POST_MQCLOSE.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQCLOSE_STRUC_ID. The constant AMPHMQCLOSE_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHMQCLOSE_VERSION_1.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.
- `pHobj (PMQHOBJ)`

  Pointer to the WebSphere MQ object handle.
- `pOptions (PMQLONG)`

  Pointer to the MQCLOSE options.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

AMPHMQCMIT (AMI C MQCMIT parameter structure)

This is used with AMIP_PRE_MQCMIT and AMIP_POST_MQCMIT.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQCMIT_STRUC_ID. The constant AMPHMQCMIT_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. Structure version number. The value must be AMPHMQCMIT_VERSION_1.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

AMPHMQCONN (AMI C MQCONN parameter structure)

This is used with AMIP_PRE_MQCONN and AMIP_POST_MQCONN.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQCONN_STRUC_ID. The constant AMPHMQCONN_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHMQCONN_VERSION_1.
- `pQMgrName (PMQCHAR)`

  Pointer to the WebSphere MQ queue manager name.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

AMPHMQCONNX (AMI C MQCONNX parameter structure)

This is used with AMIP_PRE_MQCONNX and AMIP_POST_MQCONNX.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQCONNX_STRUC_ID. The constant AMPHMQCONNX_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHMQCONNX_VERSION_1.
- `pQMgrName (PMQCHAR)`

  Pointer to the WebSphere MQ queue manager name.
- `pConnectOpts (PMQCNO)`

  Pointer to the MQCONNX options.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

AMPHMQDISC (AMI C MQDISC parameter structure)

This is used with AMIP_PRE_MQDISC and AMIP_POST_MQDISC.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQDISC_STRUC_ID. The constant AMPHMQDISC_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHMQDISC_VERSION_1.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

AMPHMQGET (AMI C MQGET parameter structure)

This is used with AMIP_PRE_MQGET and AMIP_POST_MQGET.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQGET_STRUC_ID. The constant AMPHMQGET_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHMQGET_VERSION_1.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.
- `pHobj (PMQHOBJ)`

  Pointer to the WebSphere MQ object handle.
- `pMsgDesc (PMQVOID)`

  Pointer to the WebSphere MQ message descriptor.
- `pGetMsgOpts (PMQVOID)`

  Pointer to the WebSphere MQ get message options.
- `pBufferLength (PMQLONG)`

  Pointer to buffer length.
- `ppBuffer (PPMQVOID)`

  Pointer to buffer pointer.
- `pDataLength (PMQLONG)`

  Pointer to data length.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

AMPHMQINQ (AMI C MQINQ parameter structure)

This is used with AMIP_PRE_MQINQ and AMIP_POST_MQINQ.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQINQ_STRUC_ID. The constant AMPHMQINQ_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHMQINQ_VERSION_1.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.
- `pHobj (PMQHOBJ)`

  Pointer to the WebSphere MQ object handle.
- `pSelectorCount (PMQLONG)`

  Pointer to selector count.
- `pSelectors (PMQLONG)`

  Array of selectors.
- `pIntAttrCount (PMQLONG)`

  Pointer to integer attribute count.
- `pIntAttrs (PMQLONG)`

  Array of integer attributes.
- `pCharAttrLength (PMQLONG)`

  Pointer to the length of the character attribute buffer.
- `pCharAttrs (PMQCHAR)`

  Pointer to the character attribute buffer.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

AMPHMQOPEN (AMI C MQOPEN parameter structure)

This is used with AMIP_PRE_MQOPEN and AMIP_POST_MQOPEN.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQOPEN_STRUC_ID. The constant AMPHMQOPEN_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHMQOPEN_VERSION_1.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.
- `pObjDesc (PMQVOID)`

  Pointer to the WebSphere MQ object descriptor.
- `pOptions (PMQLONG)`

  Pointer to the MQOPEN options.

- `pHobj (PMQHOBJ)`

  Pointer to the WebSphere MQ object handle.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

AMPHMQPUT (AMI C MQPUT parameter structure)

This is used with AMIP_PRE_MQPUT and AMIP_POST_MQPUT.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQPUT_STRUC_ID. The constant AMPHMQPUT_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHMQPUT_VERSION_1.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.
- `phObj (PMQHOBJ)`

  Pointer to the WebSphere MQ object handle.
- `pMsgDesc (PMQVOID)`

  Pointer to the WebSphere MQ message descriptor.
- `pPutMsgOpts (PMQVOID)`

  Pointer to the WebSphere MQ put message options.
- `pBufferLength (PMQLONG)`

  Pointer to buffer length.
- `ppBuffer (PPMQVOID`

  Pointer to buffer pointer.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

AMPHMQPUT1 (AMI C MQPUT1 parameter structure)

This is used with AMIP_PRE_MQPUT1 and AMIP_POST_MQPUT1.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQPUT1_STRUC_ID. The constant AMPHMQPUT1_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHMQPUT1_VERSION_1.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.

- `pObjDesc (PMQVOID)`

  Pointer to the WebSphere MQ object descriptor
- `pMsgDesc (PMQVOID)`

  Pointer to the WebSphere MQ message descriptor.
- `pPutMsgOpts (PMQVOID)`

  Pointer to the WebSphere MQ put message options.
- `pBufferLength (PMQLONG)`

  Pointer to buffer length.
- `ppBuffer (PPMQVOID`

  Pointer to buffer pointer.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

AMPHMQSET (AMI C MQSET parameter structure)

This is used with AMIP_PRE_MQSET and AMIP_POST_MQSET.

Fields

- `strucId (AMCHAR8)`

  Structure identifier. The value must be AMPHMQSET_STRUC_ID. The constant AMPHMQSET_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.
- `version (AMLONG)`

  Structure version number. The value must be AMPHMQSET_VERSION_1.
- `pHconn (PMQHCONN)`

  Pointer to the WebSphere MQ connection handle.
- `pHobj (PMQHOBJ)`

  Pointer to the WebSphere MQ object handle.
- `pSelectorCount (PMQLONG)`

  Pointer to selector count.
- `pSelectors (PMQLONG)`

  Array of selectors.
- `pIntAttrCount (PMQLONG)`

  Pointer to integer attribute count.
- `pIntAttrs (PMQLONG)`

  Array of integer attributes.
- `pCharAttrLength (PMQLONG)`

  Pointer to the length of the character attribute buffer.
- `pCharAttrs (PMQCHAR)`

  Pointer to the character attribute buffer.
- `pCompCode (PMQLONG)`

  Pointer to the WebSphere MQ completion code.
- `pReason (PMQLONG)`

  Pointer to the WebSphere MQ reason code.

Appendix C. Extended C AMI functions

This appendix lists the extended C AMI functions that are provided to use in policy handler libraries. These functions are supported only in C. The required constants and function prototype definitions are provided in **amtxc.h**. If required, you can also use these functions in normal applications.

The extended C AMI functions include functions to set or inquire the integer and string properties for the connection, service, message, and policy objects. Valid integer property values and ranges are listed in the following sections. All strings have a maximum length of AMSTR_MAX_LEN bytes, except for AMI object names, which have a maximum length of AMLEN_MAX_NAME_LEN. Some lengths might be restricted further when connected to WebSphere MQ.

Unless otherwise stated, all properties described in the following sections are specified through the AMI repository (where it includes a definition for the object concerned).

Connection object properties

Connection object properties can be integer properties or string properties.

Connection integer properties

AMCON_INT_TRP_TYPE

Transport type. Specifies which WebSphere MQ library the AMI dynamically loads and uses. The initial value of this property is established during object creation, before any policy handler invocation.

Valid values are:

- AMCON_TRP_TYPE_MQ_CLIENT

  Use the WebSphere MQ client library.
- AMCON_TRP_TYPE_MQ_SERVER

  Use the WebSphere MQ server library.
- AMCON_TRP_TYPE_AUTO (the default)

  Automatically determine which library to use, based on whether the WebSphere MQ server library can be located.
- AMCON_TRP_TYPE_OTHER (other)

  If the transport type is set to this value during any AMOP_CONNECTION_OPEN invocation, certain WebSphere MQ range and value limits are relaxed when other AMI object properties are set. This value can be set from a policy handler, but cannot be specified using the AMI Administration Tool.

AMCON_INT_CCSID

Coded character set identifier of the connection. This reflects the coded character set identifier of the transport connection. It provides the default CCSID value used for message element strings (including topic and filter elements) that an application program adds to, or retrieves from, a message.

This property value is established by using a WebSphere MQ MQINQ call during AMOP_CONNECTION_OPEN processing, after the queue manager connection is established. To change this value, a policy handler must set it subsequently, but before AMOP_CONNECTION_OPEN processing completes.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMCON_INT_DST_SUPPORT

Native distribution list support. This defines whether native WebSphere MQ distribution list support is available in the underlying message transport, and whether it is used.

This property value is established by using a WebSphere MQ MQINQ call during AMOP_CONNECTION_OPEN processing, after the queue manager connection is established. To ensure that a policy handler is invoked separately for each sender service in a distribution list, this MQINQ call must be intercepted, or a policy handler must subsequently set the value to AMCON_NO before AMOP_CONNECTION_OPEN processing completes.

For a policy handler that replaces the underlying transport with something other than WebSphere MQ, this property must be set to AMCON_NO during AMINV_CONNECTION_OPEN processing.

Valid values are:

- AMCON_NO

  All processing operations treat the distribution list as a list of separate sender services.
- AMCON_YES

  Distribution list operations attempt to exploit native WebSphere MQ distribution list support.

AMCON_INT_TRUSTED

Transport connection is trusted. To change this value, a policy handler must set it during AMINV_CONNECTION_OPEN invocation point processing.

Valid values are:

- AMCON_NO
- AMCON_YES

  Use a fast-path connection for trusted applications.

AMCON_INT_MQHOBJ

The object handle of the WebSphere MQ queue manager. This property value is established during AMOP_CONNECTION_OPEN/MQCONN(X) processing.

Valid values are:

- AMH_MIN to AMH_MAX

AMCON_INT_MQHCON

The connection handle of the WebSphere MQ queue manager. This property value is established during an AMOP_CONNECTION_OPEN or MQINQ invocation point after MQCONN(X). A value of AMH_INVALID_HCON indicates that the connection is closed.

Valid values are:

- AMH_MIN to AMH_MAX

AMCON_INT_USER_CONTEXT

The user context handle. The AMI does not use this property. It is provided so that other programs, such as a policy handler, can associate their own context information with a connection object.

Valid values are:

- AMH_MIN to AMH_MAX

Connection string properties

AMCON_STR_NAME

The name of the connection object. This property cannot be changed.

AMCON_STR_MGR_NAME

The name used to connect to the queue manager. This is obtained from the repository and local host files. It might be blank if the connection is to the default queue manager.

AMCON_STR_REAL_MGR_NAME
The name of the real queue manager to which the AMI is connected. This might be different to the value of AMCON_MGR_NAME.

This value is established during an AMOP_CONNECTION_OPEN or MQINQ invocation after MQCONN(X). It is used for certain name/value pair values in the RFH message header during the construction of a publish/subscribe message.

AMCON_STR_CLIENT_CHL_NAME
The WebSphere MQ client name of the server-connection channel. This value is used during AMOP_CONNECTION_OPEN/MQCONN(X) processing for WebSphere MQ client connections only.

AMCON_STR_SERVER_TCP_ADDR
The WebSphere MQ Client TCP address of the WebSphere MQ server. This value is used during AMOP_CONNECTION_OPEN/MQCONN(X) processing for WebSphere MQ client connections only.

Message object properties

Message object properties can be integer properties or string properties.

Message integer properties

AMMSG_INT_BACKOUT_CNT

Backout count. This specifies the number of times a message that is included in a unit-of-work is backed out during a receive.

When sending a message, this property is ignored. When receiving a message, this should be set by the message transport before calling **amxMsgUpdated**. This is equivalent to the BackoutCount field in the WebSphere MQ message descriptor.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMMSG_INT_CCSID

Coded character set identifier (CCSID). This specifies the CCSID of the message and provides the function performed by **amMsgGetCCSID** or **amMsgSetCCSID**.

If the message redefines the CCSID in one or more of its message headers, this property specifies the CCSID of the first message header.

When receiving a message, this should be set by the message transport before calling **amxMsgUpdated**. If the message transport performs data conversion, it should update this field to reflect the final CCSID of the converted message. This is equivalent to the CodedCharSetId field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_CCSID_INHERIT (Inherit)

  The CCSID value is inherited from the transport connection.
- AMMSG_CCSID_EMBEDDED (Embedded)

  All CCSID values are specified in the message data.
- AMMSG_CCSID_DFT to AMINT_PROP_MAX (that is, 0-999999999)

AMMSG_INT_MSG_DATA_LEN

Message data length (excluding message headers). This specifies the number of data bytes in the message after any RFH message headers.

When sending a message, the AMI updates this value as the application writes bytes to the message during message construction. When receiving a message, the AMI sets this value during **amxMsgUpdated** processing, when it is called by the message transport .

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (0-999999999).

AMMSG_INT_MSG_LEN

Message length (including message headers). This specifies the total length of the message, including any RFH and other message headers.

When sending a message, the AMI sets this during **amxMsgAssemble** processing when it is called by the message transport. When receiving a message, this is set by the message transport before calling **amxMsgUpdated**.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (0-999999999).

AMMSG_INT_ORIGINAL_LEN

Original length (for use with report messages). This is used only for report messages and specifies the length of the original message to which the report applies (if the original message was a segment, this is the length of the segment rather than the length of the logical message).

When sending a report message, the application can set this value. When receiving a message, the message transport sets this value before calling **amxMsgUpdated**. This is equivalent to the OriginalLen field of the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_ORIGINAL_LEN_UNDEFINED
- AMINT_PROP_MIN to AMINT_PROP_MAX (0-999999999).

AMMSG_INT_DATA_OFFSET

Data offset. This is the offset in bytes from the start of the message data (after any message headers) at which the next read or write bytes occurs.

When constructing a message for sending, the AMI updates this value as the application writes bytes to the message. When receiving a message, the message transport should set this to zero by using **amMsgReset** before calling **amxMsgUpdated**.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (0-999999999).

AMMSG_INT_INT_ENCODING

Integer encoding. This specifies the integer encoding of the message.

When sending a message, the application can set this value by using **amMsgSetEncoding**. When receiving a message, the message transport sets this value before **amxMsgUpdated** is called. If the message transport performs data conversion, it should update this field to reflect the integer encoding of the converted message. This is equivalent to the integer part of the Encoding field in the WebSphere MQ message descriptor.

Valid values are:

- AMENC_NORMAL (normal)
- AMENC_REVERSED (reversed)
- AMENC_UNDEFINED (undefined)

AMMSG_INT_DEC_ENC

Decimal encoding. This specifies the decimal encoding of the message.

When sending a message, the application can set this value by using **amMsgSetEncoding**. When receiving a message, the message transport sets this value before **amxMsgUpdated** is called. If the message transport performs data conversion, it should update this field to reflect the decimal encoding of the converted message. This is equivalent to the decimal part of the Encoding field in the WebSphere MQ message descriptor.

Valid values are:

- AMENC_NORMAL (normal)
- AMENC_REVERSED (reversed)
- AMENC_UNDEFINED (undefined)

AMMSG_INT_FLOAT_ENCODING

Float encoding. This specifies the floating point encoding of the message.

When sending a message, the application can set this value by using **amMsgSetEncoding**. When receiving a message, the message transport sets this value before **amxMsgUpdated** is called. If the message transport performs data conversion, it should update this field to reflect the floating point encoding of the converted message. This is equivalent to the floating point part of the Encoding field in the WebSphere MQ message descriptor.

Valid values are:

- AMENC_NORMAL (IEEE format)
- AMENC_REVERSED (IEEE format reversed)
- AMENC_NORMAL_FLOAT_390 (390 format)
- AMENC_UNDEFINED (undefined)

AMMSG_INT_FB

Feedback or report code. This sets the feedback code for a message and provides the function performed by **amMsgGetReportCode** or **amMsgSetReportCode**.

When receiving a message, the message transport should set this value before calling **amxMsgUpdated**. This is equivalent to the Feedback field in the WebSphere MQ message descriptor.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMMSG_INT_EXPIRY

Expiry interval. This specifies the expiry interval in (tenths of a second) of the message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport sets this value before **amxMsgUpdated** is called. This is equivalent to the Expiry field in the WebSphere MQ message descriptor.

Valid values are:

- AMEI_UNLIMITED
- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMMSG_INT_MSG_TYPE

Message type. This sets the message type of a message and provides the function performed by **amMsgGetType** or **amMsgSetType**.

When receiving a message, the message transport should set this value before calling **amxMsgUpdated**. This is equivalent to the MsgType field in the WebSphere MQ message descriptor.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999).

  This includes the following predefined values:

  - AMMT_NONE (None)
  - AMMT_REQUEST (Request)
  - AMMT_REPLY (Reply)
  - AMMT_REPORT (Report)
  - AMMT_DATAGRAM (Datagram)

AMMSG_INT_SEQ_NO
Sequence number. This specifies the sequence number of a message within a message group.

When sending a message, the message transport generates this information automatically, and this property is ignored. When receiving a message, the message transport should set this value before calling **amxMsgUpdated**. This is equivalent to the MsgSeqNumber field in the WebSphere MQ message descriptor.

Valid values are:

- 1 to AMINT_PROP_MAX (that is, 1-999999999)

AMMSG_INT_PERSISTENT
Persistence. This specifies the persistence of the message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the Persistence field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_NO (Not persistent)
- AMMSG_YES (Persistent)

AMMSG_INT_PRIORITY
Priority. This specifies the priority of the message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the Priority field in the WebSphere MQ message descriptor.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMMSG_INT_RO_EXCEPTION
Report option: exception report messages. This specifies whether or not exception report messages are required for this message.

When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the Exception report options part of the Report field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_RO_EXCPTN_NO (Not required)
- AMMSG_RO_EXCPTN_YES (Required)
- AMMSG_RO_EXCPTN_DATA (Required with data)
- AMMSG_RO_EXCPTN_FULL (Required with full data)

AMMSG_INT_RO_EXPIRY
Report option: expiry report messages. This specifies whether or not expiry report messages are required for this message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the Expiry report options part of the Report field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_RO_EXPIRY_NO (Not required)
- AMMSG_RO_EXPIRY_YES (Required)
- AMMSG_RO_EXPIRY_DATA (Required with data)
- AMMSG_RO_EXPIRY_FULL (Required with full data)

AMMSG_INT_RO_COA

Report option: confirm-on-arrival report messages. This report option specifies whether or not confirm-on-arrival report messages are required for this message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the COA report options part of the Report field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_RO_COA_NO (Not required)
- AMMSG_RO_COA_YES (Required)
- AMMSG_RO_COA_DATA (Required with data)
- AMMSG_RO_COA_FULL (Required with full data)

AMMSG_INT_RO_COD

Report option: confirm-on-delivery report messages. This specifies whether or not confirm-on-delivery report messages are required for this message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the COD report options part of the Report field in the WebSphere MQ message descriptor

Valid values are:

- AMMSG_RO_COD_NO (Not required)
- AMMSG_RO_COD_YES (Required)
- AMMSG_RO_COD_DATA (Required with data)
- AMMSG_RO_COD_FULL (Required with full data)

AMMSG_INT_RO_COPY_MSG_ID

Report option: copy MessageId (to CorrelId). This specifies the whether or not the MessageId (rather than the CorrelId) from messages that are sent using this policy is copied to the CorrelId of report (or reply) messages that are sent in response.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the Copy MessageId or CorrelId report options part of the Report field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_NO
- AMMSG_YES

AMMSG_INT_RO_DISCARD

Report option: disposition. This specifies the whether or not to discard messages that are sent using this policy and that cannot be delivered, rather than put them on the dead letter queue.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the Dead Letter Queue or Discard report options part of the Report field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_NO
- AMMSG_YES

AMMSG_INT_RO_PAN

Report option: positive-action-notification report messages. This specifies whether or not PAN report messages are required for this message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the PAN report options part of the Report field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_NO (Not required)
- AMMSG_YES (Required)

AMMSG_INT_RO_NAN

Report option: negative-action-notification report messages. This specifies whether or not NAN report messages are required for this message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the NAN report options part of the Report field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_NO (Not required)
- AMMSG_YES (Required)

AMMSG_INT_RO_NEW_MSG_ID

Report option: New MessageId. When sending a message, this specifies whether to generate a new MessageId for a report (or reply) message sent in response, or whether to use the MessageId from the original message.

When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the New MessageId or Pass MessageId report options part of the Report field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_NO
- AMMSG_YES

AMMSG_INT_SEGMENTS

Segmentation allowed. This specifies whether or not the message transport can optionally split this message into segments during a send.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the Segmentation Allowed option of the MsgFlags field in the WebSphere MQ message descriptor.

Valid values are:

- AMMSG_NO
- AMMSG_YES

AMMSG_INT_IN_GROUP

The message is a member of a group. This specifies whether or not the message is a member of a message group, and provides the function performed by **amMsgGetGroupStatus** or **amMsgSetGroupStatus**.

When sending a message, the application can set this value by. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the Segmentation Allowed option of the MsgFlags field in the WebSphere MQ message descriptor.

Valid values are:

- AMGRP_MSG_NOT_IN_GRP
- AMGRP_FIRST_MSG_IN_GRP
- AMGRP_MIDDLE_MSG_IN_GRP
- AMGRP_LAST_MSG_IN_GRP
- AMGRP_ONLY_MSG_IN_GRP

AMMSG_INT_APPL_CCSID

Application coded character set identifier (CCSID). This specifies the CCSID that the application uses to add or retrieve message element data to or from the message, and provides the function performed by **amMsggetElementCCSID** or **amMsgSetElementCCSID**. The application can set this value. By default, the application uses the message transport CCSID obtained from the connection.

Valid values are:

- AMMSG_CCSID_DFT_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMMSG_INT_USER_CONTEXT

User context handle. The AMI does not use this property. It is provided so that other programs, such as a policy handler, can associate their own context information with a policy object.

Valid values are:

- AMH_MIN to AMH_MAX

Message string properties

AMMSG_STR_NAME
Name of the message object. This provides the function performed by **amMsgGetName**. This property cannot be changed.

AMMSG_STR_FORMAT
Format of the message. This specifies the message format and provides the function performed by **amMsgGetFormat** or **amMsgSetFormat**.

When sending a message, the application can set this value. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the Format field in the WebSphere MQ message descriptor.

AMMSG_STR_CORREL_ID
Correlation identifier. This specifies the CorrelId and provides the function performed by **amMsgGetCorrelId** or **amMsgSetCorrelId**.

When sending a message, the application can set this value or, using the appropriate policy option, the message transport can generate this value automatically. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the CorrelId field in the WebSphere MQ message descriptor.

AMMSG_STR_MSG_ID
Message identifier. This specifies the MessageId and provides the function performed by **amMsgGetMsg**.

When sending a message, either the message transport generates this value automatically, or the AMI sets this value (using information from a request message). When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the MsgId field in the WebSphere MQ message descriptor.

AMMSG_STR_GROUP_ID
Message group identifier. This specifies the message Group Id.

When sending a message, either the message transport generates this value automatically, or the AMI sets this value (using information from a request message). When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the MsgId field in the WebSphere MQ message descriptor.

Policy object properties

Policy object properties can be integer properties or string properties.

Policy integer properties

AMPOL_INT_SYNCPOINT
Sync point selected. This specifies whether or not messages sent and received using this policy are included in a unit-of-work.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_CONTEXT
Message context. This controls the level of message context information from a message receiver that uses this policy that is passed on in an output message that is sent using this policy.

Valid values are:

- AMPOL_CONTEXT_NONE (No context)
- AMPOL_CONTEXT_AS_TRP (As transport)
- AMPOL_CONTEXT_PASS_ID (Pass identity context)
- AMPOL_CONTEXT_PASS_ALL (Pass all context)

AMPOL_INT_SND_IMPL_OPEN
Implicitly open sender. This specifies whether a sender service should be implicitly opened to complete a send operation successfully using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_SND_LEAVE_OPEN
Leave sender open. This specifies whether an implicitly opened sender service remains open after a send operation using this policy has successfully completed.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_PRIORITY
Message priority. This specifies the priority of messages sent using this policy.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMPOL_INT_PERSISTENCE
Message persistence. This specifies the persistence of messages sent using this policy.

Valid values are:

- AMPOL_PERSISTENCE_AS_TRP (as transport)
- AMPOL_PERSISTENCE_NO (not persistent)
- AMPOL_PERSISTENCE_YES (persistent)

AMPOL_INT_EXPIRY
Expiry interval. This specifies the expiry interval (in tenths of a second) of messages sent using this policy.

Valid values are:

- AMEI_UNLIMITED
- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMPOL_INT_RETRY_COUNT
Retry count. This specifies the number of retries to use when messages that are sent using this policy encounter a temporary error condition.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMPOL_INT_RETRY_INTERVAL
Retry interval. This specifies the time interval (in milliseconds) between retries when messages that are sent using this policy encounter a temporary error condition.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMPOL_INT_REPORT_DATA
Data required in report messages. This specifies the amount of data, if any, required in report messages that are sent in response to messages that are sent using this policy.

Valid values are:

- AMPOL_REPORT_DATA_NO (no data)
- AMPOL_REPORT_DATA_YES (data)
- AMPOL_REPORT_DATA_FULL (full data)

AMPOL_INT_RO_EXCEPTION
Report option: exception report messages required. This specifies whether or not exception report messages are required for messages that are sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_RO_EXPIRY
Report option: expiry report messages required. This specifies whether or not expiry report messages are required for messages that are sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_RO_COA
Report option: confirm-on-arrival report messages required. This specifies whether or not confirm-on-arrival report messages are required for messages that are sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_RO_COD

Report option: confirm-on-delivery report messages required. This report option specifies whether or not confirm-on-delivery report messages are required for messages that are sent using this policy.

Valid values are:
- AMPOL_NO
- AMPOL_YES

AMPOL_INT_RO_COPY_MSG_ID

Report option: copy MessageId to CorrelId. This specifies the whether or not to copy the MessageId (rather than CorrelId) from messages that are sent using this policy to the CorrelId of report or reply messages that are sent in response.

Valid values are:
- AMPOL_NO
- AMPOL_YES

AMPOL_INT_RO_DISCARD

Report option: disposition. This specifies the whether or not messages that are sent using this policy are discarded (rather than put to the dead letter queue) if they cannot be delivered.

Valid values are:
- AMPOL_NO
- AMPOL_YES

AMPOL_INT_NEW_CORREL_ID

Generate new CorrelId. This report option specifies whether or not to generate a new CorrelId automatically for messages that are sent using this policy (except when sending a report or response message where the MessageId or CorrelId from the request message is always used).

Valid values are:
- AMPOL_NO
- AMPOL_YES

AMPOL_INT_SEGMENT

Segment message. This specifies whether or not messages that are sent using this policy that can be segmented for transmission by the underlying message transport.

Valid values are:
- AMPOL_NO
- AMPOL_YES

AMPOL_INT_SPLIT_LOGICAL

Split files on logical boundaries. This specifies whether or not files that are sent using this policy are split into separate messages on logical, rather than physical, boundaries, as determined by the file record length.

On Windows, HP-UX, AIX, and Sun Solaris, if the File Record Length is zero, this is the end of a line. On z/OS, this is a record boundary.

If physical splitting is used, files are split into separate messages on boundaries that the AMI determines, and the messages include sufficient data to recreate them (as files) at the receiver.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_FILE_RCD_LEN

File record length. This specifies the boundary used with logical file splitting to split a file into separate messages.

On Windows, HP-UX, AIX, and Sun Solaris, a value of zero specifies that the end of a line is used. On z/OS, a value of zero specifies that the record boundary is used.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMPOL_INT_BIND_ON_OPEN

Bind on open. This specifies when a sender service binds to its underlying message queue in a cluster environment.

Valid values are:

- AMPOL_BIND_ON_OPEN_AS_TRP (As transport)
- AMPOL_BIND_ON_OPEN_NO (Do not bind on open)
- AMPOL_BIND_ON_OPEN_YES (Bind on open)

AMPOL_INT_RCV_IMPL_OPEN

Implicitly open receiver. This specifies whether or not to open a receiver service implicitly in order to complete a receive operation successfully using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_RCV_LEAVE_OPEN

Leave receiver open. This specifies whether or not to leave an implicitly opened receiver service open after a receive operation using this policy has completed successfully.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_RCV_CLOSE_DELETE

Delete temporary dynamic queue on close. This specifies whether or not to delete a temporary dynamic queue when its receiver service is closed using this policy.

Valid values are:

- AMPOL_CLOSE_DELETE_NO

  Do not delete the temporary dynamic queue.
- AMPOL_CLOSE_DELETE_YES

  Delete the temporary dynamic queue if it is empty.
- AMPOL_CLOSE_DELETE_PURGE

  If the temporary dynamic queue is not empty, discard its messages, the delete it.

AMPOL_INT_WAIT_INTERVAL
Wait interval. This specifies the time (in milliseconds) to wait when receiving a message using this policy.

Valid values are:

- AMWT_UNLIMITED (unlimited)
- AMINT_PROP_MIN-AMINT_PROP_MAX (that is, 0-999999999)

AMPOL_INT_WAIT_INTERVAL_RO
Wait interval is read-only. This specifies whether or not the wait interval value for this policy is read-only and cannot be changed.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_CONVERT
Convert message data on receive. This specifies whether or not to perform code page and encoding conversion on messages received using this policy (target code page and encoding values for the conversion are specified by the selection message).

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_WAIT_WHOLE_GROUP
Wait for whole message group. This specifies whether or not to wait for all the messages in a group to become available before returning a message from the group, when receiving a message using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_HANDLE_POISON_MSG
Handle poison messages. This specifies whether or not to perform poison message handling when receiving a message using this policy (see "Note 1" on page 501for a description of poison message handling).

Valid values are:

AMPOL_NO
AMPOL_YES

AMPOL_INT_ACCEPT_TRUNCATED
Accept truncated messages. This specifies whether or not to accept truncated messages for a message that is too large for the application message buffer when receiving a message using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_SHARED
Open shared. This specifies whether or not to open the underlying message queue as a shared queue (rather than an exclusive queue) when opening a receiver using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_FILE_DISP

File disposition. This specifies whether to create a new file, or whether to overwrite or append to an existing file, when receiving a file using this policy.

Valid values are:

- AMPOL_FILE_DISP_NEW
- AMPOL_FILE_DISP_OVERWRITE
- AMPOL_FILE_DISP_APPEND

AMPOL_INT_RETAIN_PUBS

Retain publications. This specifies whether or not the broker retains publications when they are published using this policy.

Valid values are:

AMPOL_NO
AMPOL_YES

AMPOL_INT_PUB_OTHERS_ONLY

Publish to others only. This specifies whether or not to suppress sending publications back to the publisher (if it is registered as a subscriber) when published using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_SUPPRESS_REG

Suppress implicit publisher registration. This specifies whether or not to suppress implicit publisher registration for publications that are sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_PUB_LOCAL

Publish locally. This specifies whether or not publications are sent only to subscribers that are local to the broker when publications are sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_ACCEPT_DIRECT

Accept direct requests. This specifies whether or not publishers accept direct requests from subscribers when registration results from a publication that is sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_PUB_ANON

Publisher registration is anonymous. This specifies whether or not publisher registration is anonymous when registration results from a publication that is sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_PUB_USE_CORREL_ID

Use CorrelId as publisher Id. This specifies that the broker uses the CorrelId of the message as part of the publisher's identity when publisher registration results from a publication that is sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_SUB_LOCAL

Subscribe locally. This specifies whether or not the subscriber is sent only publications published with the local option when it subscribes using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_NEW_PUBS_ONLY

Send new publications only. This specifies whether or not a subscriber is sent only new publications when it subscribes using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_PUB_ON_REQ

Publish on request only. This specifies whether or not the subscriber is only sent retained publications on sending a request update when it subscribes using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_INFORM_IF_RET

Inform if retained. This specifies whether or not the broker informs the subscriber whether a publication is retained when it subscribes using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_REMOVE_ALL_SUBS

Remove all subscriptions. This specifies whether or not the broker removes all subscriptions for this subscriber when it unsubscribes using this policy.

Valid values are:
- AMPOL_NO
- AMPOL_YES

AMPOL_INT_SUB_ANON
Subscriber registration is anonymous. This specifies whether or not subscribers remain anonymous when they subscribe using this policy.

Valid values are:
- AMPOL_NO
- AMPOL_YES

AMPOL_INT_SUB_USE_CORREL_ID
Use CorrelId as subscriber Id. This specifies that the broker uses the CorrelId of the message as part of the subscriber's identity when it subscribes using this policy.

Valid values are:
- AMPOL_NO
- AMPOL_YES

AMPOL_INT_SUB_DLY_PERSISTENCE
Delivery persistence of publications. This specifies whether or not publications that the broker sends to subscribers are persistent when they subscribe using this policy.

Valid values are:
- AMPOL_PERSISTENCE_AS_TRP (as transport)
- AMPOL_PERSISTENCE_NO (not persistent
- AMPOL_PERSISTENCE_YES (persistent)

AMPOL_INT_USER_CONTEXT
User context handle. The AMI does not use this property. It is provided so that other programs, such as a policy handler, can associate their own context information with a policy object.

Valid values are:
- AMH_MIN to AMH_MAX

Policy string properties

AMPOL_STR_NAME
The name of the policy object. This provides the function performed by **amPolGetNam**e. This property cannot be changed.

AMPOL_STR_APPL_GROUP
Application group. This specifies the application group name when sending to a WebSphere MQ Integrator Version 1 broker using this policy.

AMPOL_STR_SUBS_POINT
Subscription point. This specifies the subscription point when subscribing to a WebSphere MQ Integrator Version 2 broker using this policy.

Service object properties

Service object properties can be integer properties or string properties.

Service integer properties

AMSRV_INT_TYPE
Service type. This specifies the type of sender or receiver service. This property value is established at object creation time, based on the type of service object being created and its repository definition (if one exists).

Valid values are:

- AMSRV_TYPE_RCV (receiver)
- AMSRV_TYPE_SND (sender)
- AMSRV_TYPE_DST_SND (distribution list sender)
- AMSRV_TYPE_PUB_SND (publisher sender)
- AMSRV_TYPE_SUB_SND (subscriber sender)
- AMSRV_TYPE_SUB_RCV (subscriber receiver)

AMSRV_INT_SND_TYPE
Sender type. This property is meaningful only for sender services and specifies the sender type. This controls the type and content of Rules and Formatting (RFH) message headers added by the AMI.

Valid values are:

- AMSRV_SND_TYPE_NATIVE (Native - default value)

  Name/value elements are included in an MQRFH header.
- AMSRV_SND_TYPE_MQINT_V1 (WebSphere MQ Integrator V1)

  Name/value elements are included in an MQRFH header, together with OPT_APP_GROUP and OPT_MSG_TYPE elements, where appropriate.
- AMSRV_SND_TYPE_MQINT_V2 (WebSphere MQ Integrator V2)

  Name/value elements are included in an MQRFH2 header, together with Default MCD Domain, Default MCD Format, Default MCD Type, Default MCD Format, Delivery Persistence, and Subscription Point elements, where appropriate.
- AMSRV_SND_TYPE_RFH1 (RFH1)

  Name/value elements are included in an MQRFH header.
- AMSRV_SND_TYPE_RFH2 (RFH2)

  Name/value elements are included in an MQRFH2 header.

AMSRV_INT_DEFN_TYPE
Definition type. This specifies the definition type and provides the function performed by **amRcvGetDefnType** or **amSubGetDefnType**.

This property value is initially set at object creation time to Dynamic, Predefined, or Undefined (the default), based on the repository definition (if one exists). When the service is opened, Undefined or Dynamic may change to Dynamic or Temporary dynamic, based on the attributes of the underlying queue. If a service is used (that is, specified) as the response sender service when receiving a request message, Undefined changes to Response.

Valid values are:
- AMDT_UNDEFINED (Undefined)
- AMDT_RESPONSE (Response)
- AMDT_TEMP_DYNAMIC (Temporary dynamic)
- AMDT_DYNAMIC (Dynamic)
- AMDT_PREDEFINED (Predefined)

AMSRV_INT_ENCODING

Encoding. This provides the function performed by **amSndGetEncoding**, **amPubGetEncoding**, or **amSubGetEncoding**.

This is applicable to sender services with a destination application that cannot perform data conversion for the required encoding. This specifies the encoding of the destination application for this service and exists to provide AMI applications with the information they need to correctly convert message data to the required encoding before it is sent. This property value is set at object creation time based on the repository definition (if one exists).

Valid values are:
- AMENC_NORMAL (Normal)
- AMENC_REVERSED (Reversed)
- AMENC_NORMAL_FLOAT_390 (Normal 390 floating point)
- AMENC_REVERSED_FLOAT_390 (Reversed, 390 floating point)
- AMENC_UNDEFINED (Undefined - default)

AMSRV_INT_CCSID

Coded character set identifier. This provides the function performed by **amSndGetCCSID**, **amPubGetCCSID**, or **amSubGetCCSID**.

This is applicable to sender services with a destination application that cannot perform data conversion for the required CCSID. This specifies the CCSID of the destination application for this service and provides AMI applications with the information needed to correctly convert message data to the required CCSID before it is sent. This property value is set at object creation time based on the repository definition (if one exists).

Valid values are:
- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMSRV_INT_NEXT_SND_HANDLE

Next sender service handle (in a distribution list). This provides the function performed by **amDstGetSenderHandle**.

This property value is established from the chain of sender services that represent a distribution list when the distribution list is created. It specifies the handle of the next sender service after this in the distribution list. For the last sender service in a distribution list, this value is set to AMH_NULL_HANDLE.

Valid values are:
- AMH_MIN to AMH_MAX

AMSRV_INT_SUB_RCV_HANDLE
Subscriber receiver handle. This is applicable only for the sender service of a subscriber.

This property value is established when the subscriber is created and specifies the handle of the receiver service for the subscriber.

Valid values are:

- AMH_MIN to AMH_MAX

AMSRV_INT_MQHOBJ
The object handle of the WebSphere MQ queue. This property value is established during AMOP_SENDER_OPEN/MQOPEN or AMOP_RECEIVER_OPEN/MQOPEN processing. A value of AMH_INVALID_HOBJ indicates that the service is closed.

Valid values are:

- AMH_MIN to AMH_MAX

AMSRV_INT_USER_CONTEXT
The user context handle. The AMI does not use this property. It is provided so that other programs, such as a policy handler, can associate their own context information with a service object.

Valid values are:

- AMH_MIN to AMH_MAX

Service string properties

AMSRV_STR_NAME
The name of the sender or receiver service object. This provides the function performed by **amSndGetName**, **amRcvgetName**, **amPubGetname**, or **amSubGetName**. This property cannot be changed.

AMSRV_STR_Q_NAME
Queue name. This provides the function performed by **amRcvGetQueueName** or **amSubGetQueueName**.

If the definition type is Predefined, this property value is established at object creation time based on the repository definition (if one exists). If the definition type is Dynamic (or Temporary dynamic), it is determined from the name of the underlying dynamic queue that is created when the service is opened. If the definition type is response, it is determined from the name of reply-to queue when receiving a request message.

AMSRV_STR_MGR_NAME
Queue manager name. This is the name of the queue manager where the queue resides. A null string indicates that the queue manager is that to which the AMI session is connected.

If the definition type is Predefined, this property value is established at object creation time based on the repository definition (if one exists). If the definition type is response, it is determined from the name of reply-to queue manager when receiving a request message.

AMSRV_STR_MODEL_Q_NAME
Model queue name. This specifies name of the model queue to be used when opening a service of definition type Dynamic.

AMSRV_STR_DFT_MSG_FORMAT
Default message format. This specifies the default message format to be used if the message format is AMFMT_NONE. It also specifies the value of the AMPS_MSG_TYPE (OPT_MSG_TYPE) element value with sender type WebSphere MQ Integrator V1.

AMSRV_STR_DYNAMIC_PREFIX
Model queue name. This specifies the prefix used for the name of dynamic queue that is created when opening a service of definition type Dynamic.

AMSRV_STR_DFT_MCD_DOMAIN
Default MCD domain. This specifies the AMMCD_MSG_SERVICE_DOMAIN (mcd.Msd) element value with sender type WebSphere MQ Integrator V2.

AMSRV_STR_DFT_MCD_SET
Default MCD set. This specifies the AMMCD_MSG_SERVICE_DOMAIN (mcd.Msd) element value with sender type WebSphere MQ Integrator V2.

AMSRV_STR_DFT_MCD_TYPE
Default MCD type. This specifies the AMMCD_MSG_SERVICE_DOMAIN (mcd.Msd) element value with sender type WebSphere MQ Integrator V2.

AMSRV_STR_DFT_MCD_FORMAT
Default MCD format. This property value is established at object creation time, based on the repository definition (if one exists). It specifies the AMMCD_MSG_SERVICE_DOMAIN (mcd.Msd) element value with sender type WebSphere MQ Integrator V2.

AMSRV_STR_CUST_PARMS
Custom parameters. This specifies the custom parameters defined in the repository for the service point and exists so that a service point can be customized is a way that is meaningful to a policy handler.

Connection object functions

amxConSetStringProp

Sets the specified string property in the connection object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxConSetStringProp(
  AMHCON    hCon,
  AMLONG    propId,
  AMLONG    propLen,
  AMSTR     pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hCon	The handle of the connection object that amSesOpen creates (input).
propId	The property identifier (input). See “Connection string properties” on page 593 for a list of valid values (you cannot set AMCON_STR_NAME).
propLen	The property length (input). If set to AMLEN_NULL_TERM, the property is a null-terminated string.
pProp	The property string (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amxConGetStringProp

Returns the specified string property of the connection object.

```
AMBOOL amxConGetStringProp(
  AMHCON    hCon,
  AMLONG    propId,
  AMLONG    buffLen,
  PAMLONG   pPropLen,
  AMSTR     pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);syntax
```

hCon	The handle of the connection object created during amSesOpen (input).
propId	The property identifier (input). See “Connection string properties” on page 593 for a list of valid values.
buffLen	The length of the buffer specified by pProp (input). If this is set to zero, the property string is not returned.
pPropLen	The property length excluding any terminating null (output). If this is set to NULL, the length is not returned.
pProp	The property string (output). Any bytes in the buffer after the property string are set to null, up to the specified buffer length or property length, whichever is smaller.
pCompCode	Completion code (output).
pReason	Reason code (output).

amxConSetIntProp

Sets the specified integer property in the connection object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxConSetIntProp(
  AMHCON    hCon,
  AMLONG    propId,
  AMLONG    prop,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hCon	The handle of the connection object created during amSesOpen (input).
propId	The property identifier (input). See “Connection integer properties” on page 592 for a list of valid values.
prop	The property value (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amxConGetIntProp

Returns the specified integer property of the connection object.

```
AMBOOL amxConGetIntProp(
  AMHCON    hCon,
  AMLONG    propId,
  PAMLONG   pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hCon	The handle of the connection object created during amSesOpen (input).
propId	The property identifier (input). See “Connection integer properties” on page 592 for a list of valid values.
pProp	The property value (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

Message object functions

amxMsgSetStringProp

Sets the specified string property in the message object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxMsgSetStringProp(
  AMHMSG    hMsg,
  AMLONG    propId,
  AMLONG    propLen,
  AMSTR     pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle that amSesCreateMessage returns (input).
propId	The property identifier (input). See “Message string properties” on page 602 for a list of valid values (you cannot set AMMSG_STR_NAME).
propLen	The property length (input). If set to AMLEN_NULL_TERM, the property is a null-terminated string.
pProp	The property string (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amxMsgGetStringProp

Returns the specified string property of the message object.

```
AMBOOL amxMsgGetStringProp(
  AMHMSG    hMsg,
  AMLONG    propId,
  AMLONG    buffLen,
  PAMLONG   pPropLen,
  AMSTR     pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle that amSesCreateMessage returns (input).
propId	The property identifier (input). See “Message string properties” on page 602 for a list of valid values.
buffLen	The length of the buffer specified by pProp (input). If this is set to zero, the property string is not returned.
pPropLen	The property length excluding any terminating null (output). If this is set to NULL, the length is not returned.
pProp	The property string (output). Any bytes in the buffer after the property string are set to null, up to the specified buffer length or property length, whichever is smaller.
pCompCode	Completion code (output).
pReason	Reason code (output).

amxMsgSetIntProp

Sets the specified integer property in the message object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxMsgSetIntProp(
  AMHMSG    hMsg,
  AMLONG    propId,
  AMLONG    prop,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle that amSesCreateMessage returns (input).
propId	The property identifier (input). See “Message integer properties” on page 595 for a list of valid values.
prop	The property value (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amxMsgGetIntProp

Returns the specified integer property of the message object.

```
AMBOOL amxMsgGetIntProp(
  AMHMSG    hMsg,
  AMLONG    propId,
  PAMLONG   pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hMsg	The message handle that amSesCreateMessage returns (input).
propId	The property identifier (input). See “Message integer properties” on page 595 for a list of valid values.
pProp	The property value (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amxMsgAssemble

Assembles a message in wire format with all the required headers (MQRFH message element headers and external headers), ready for transmission. The message data can be provided either by the message object, or from an external data buffer. An optional external message header can be inserted at the start of message. An additional option allows a message header to be inserted at the start of a previously assembled message.

This function enables a policy handler to replace the existing WebSphere MQ message transport. It is called from a message transport routine before a send function.

This function returns the address of the buffer containing the newly assembled wire format message. The length is determined using **amxMsgGetIntProps** with property identifier AMMSG_INT_MSG_LEN.

```
AMBOOL amxMsgAssemble(
  AMHMSG      hMsg,
  AMLONG      options,
  AMLONG      extDataLen,
  PAMBYTE     pExtData,
  AMLONG      extHdrLen,
  PAMBYTE     pExtHdr,
  PPAMBYTE    pBufferAddr,
  PAMLONG     pCompCode,
  PAMLONG     pReason);
```

hMsg The message handle that amSesCreateMessage returns (input).

options Options (input). Valid values are:

AMMSG_AMO_NONE
Assemble the message in the default way.

AMMSG_AMO_RFH
Include the message elements in an MQRFH header.

AMMSG_AMO_RFH2
Include the message elements in an MQRFH2 header.

AMMSG_AMO_EXT_HDR
Prefix an external header, but do not change the assembled message.

extDataLen Length of data in external buffer (input).

If the data is contained in the message object and there is no external data buffer, set this to zero.

pExtData Pointer to data in an external buffer (input).

If the data is contained in the message object and there is no external data buffer, set this to NULL.

extHdrLen External header length (input).

If there is no external header, set this to zero.

pExtHdr Pointer to an external header to add to the start of the message (input).

If there is no external header, set this to NULL.

pBufferAddr Pointer to the assembled message data (output).

pCompCode Completion code (output).

pReason Reason code (output).

amxMsgAllocateMem

Allocates space in the message object when receiving a message directly into the message object, rather than into an external buffer.

This function enables a policy handler to replace the existing WebSphere MQ message transport. It is called from a message transport routine before a receive function.

```
AMBOOL amxMsgAllocateMem(
  AMHMSG      hMsg,
  AMLONG      minLen,
  PAMLONG     pBuffLen,
  PPAMBYTE    pBuffAddr,
  PAMLONG     pCompCode,
  PAMLONG     pReason);
```

hMsg	The message handle that amSesCreateMessage returns (input).
minLen	The minimum required length in bytes (input). A value of zero specifies the default size.
pBuffLen	Length allocated in bytes (output).
pBuffAddr	Address of allocated buffer (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amxMsgUpdated

Parses a newly received message so that any RFH name/value elements can be extracted from the message data into an internal element table. The message data can be contained in the message object, or in an external data buffer.

This function enables a policy handler to replace the existing WebSphere MQ message transport. It is called from a message transport routine after a message is received and all required message properties are updated.

```
AMBOOL amxMsgUpdated(
  AMHMSG      hMsg,
  AMLONG      extMsgLen,
  PAMBYTE     pExtMsg,
  PAMLONG     pCompCode,
  PAMLONG     pReason);
```

hMsg	The message handle that amSesCreateMessage returns (input).
extMsgLen	The length of the message data in an external buffer property identifier (input). If the data is received into the message object and there is no external data buffer, set this to zero.
pExtMsg	Pointer to the message data in an external buffer property identifier (input). If the data is received into the message object and there is no external data buffer, set this to NULL.
pCompCode	Completion code (output).
pReason	Reason code (output).

Policy object functions

amxPolSetStringProp

Sets the specified string property in the policy object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxPolSetStringProp(
  AMHPOL    hPolicy,
  AMLONG    propId,
  AMLONG    propLen,
  AMSTR     pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPolicy	The policy handle that amSesCreatePolicy returns (input).
propId	The property identifier (input). See "Policy string properties" on page 610 for a list of valid values (you cannot set AMPOL_STR_NAME).
propLen	The property length (input). If set to AMLEN_NULL_TERM, the property is a null-terminated string.
pProp	The property string (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amxPolGetStringProp

Returns the specified string property of the policy object.

```
AMBOOL amxPolGetStringProp(
   AMHPOL    hPolicy,
   AMLONG    propId,
   AMLONG    buffLen,
   PAMLONG   pPropLen,
   AMSTR     pProp,
   PAMLONG   pCompCode,
   PAMLONG   pReason);
```

hPolicy	The policy handle that amSesCreatePolicy returns (input).
propId	The property identifier (input). See "Policy string properties" on page 610 for a list of valid values.
buffLen	The length of the buffer specified by pProp (input). If this is set to zero, the property string is not returned.
pPropLen	The property length excluding any terminating null (output). If this is set to NULL, the length is not returned.
pProp	The property string (output). Any bytes in the buffer after the property string are set to null, up to the specified buffer length or property length, whichever is smaller.
pCompCode	Completion code (output).
pReason	Reason code (output).

amxPolSetIntProp

Sets the specified integer property in the policy object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxPolSetIntProp(
  AMHPOL    hPolicy,
  AMLONG    propId,
  AMLONG    prop,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPolicy	The policy handle that amSesCreatePolicy returns (input).
propId	The property identifier (input). See “Policy integer properties” on page 603 for a list of valid values.
prop	The property value (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amxPolGetIntProp

Returns the specified integer property of the policy object.

```
AMBOOL amxPolGetIntProp(
  AMHPOL    hPolicy,
  AMLONG    propId,
  PAMLONG   pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hPolicy	The policy handle that amSesCreatePolicy returns (input).
propId	The property identifier (input). See “Policy integer properties” on page 603 for a list of valid values.
pProp	The property value (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

Service object functions

amxSrvSetStringProp

Sets the specified string property in the service object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxSrvSetStringProp(
  AMHSRV    hSrv,
  AMLONG    propId,
  AMLONG    propLen,
  AMSTR     pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSrv	The service handle that amSesCreateSender, amSesCreateReceiver, amSesCreatePublisher, or amSesCreateSubscriber returns (input).
propId	The property identifier (input). See “Service string properties” on page 613 for a list of valid values (you cannot set AMSRV_STR_NAME).
propLen	The property length (input). If set to AMLEN_NULL_TERM, the property is a null-terminated string.
pProp	The property string (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amxSrvGetStringProp

Returns the specified string property of the service object.

```
AMBOOL amxSrvGetStringProp(
  AMHSRV    hSrv,
  AMLONG    propId,
  AMLONG    buffLen,
  PAMLONG   pPropLen,
  AMSTR     pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSrv	The service handle that amSesCreateSender, amSesCreateReceiver, amSesCreatePublisher, or amSesCreateSubscriber returns (input).
propId	The property identifier (input). See “Service string properties” on page 613 for a list of valid values.
buffLen	The length of the buffer specified by pProp (input). If this is set to zero, the property string is not returned.
pPropLen	The property length excluding any terminating null (output). If this is set to NULL, the length is not returned.
pProp	The property string (output). Any bytes in the buffer after the property string are set to null, up to the specified buffer length or property length, whichever is smaller.
pCompCode	Completion code (output).
pReason	Reason code (output).

amxSrvSetIntProp

Sets the specified integer property in the service object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxSrvSetIntProp(
  AMHSRV    hSrv,
  AMLONG    propId,
  AMLONG    prop,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSrv	The service handle that amSesCreateSender, amSesCreateReceiver, amSesCreatePublisher, or amSesCreateSubscriber returns (input).
propId	The property identifier (input). See “Service integer properties” on page 611 for a list of valid values.
prop	The property value (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

amxSrvGetIntProp

Returns the specified integer property of the service object.

```
AMBOOL amxSrvGetIntProp(
  AMHSRV    hSrv,
  AMLONG    propId,
  PAMLONG   pProp,
  PAMLONG   pCompCode,
  PAMLONG   pReason);
```

hSrv	The service handle that amSesCreateSender, amSesCreateReceiver, amSesCreatePublisher, or amSesCreateSubscriber returns (input).
propId	The property identifier (input). See “Service integer properties” on page 611 for a list of valid values.
pProp	The property value (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

Appendix D. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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 United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

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 Programming License Agreement, or any equivalent agreement between us.

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.

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. 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 IBM's application programming interfaces.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

The following permission statements and conditions only apply to the XML library files (AMTXML, amtxml310, libamtxml310, and libamtxml310_r) built using software from Apache Software Foundation, and not to any code developed solely by IBM Corporation. They do not invalidate any of the terms of the IBM International Program License Agreement for this IBM product.

Copyright © 1999-2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: “This product includes software developed by the Apache Software Foundation (http://www.apache.org/).” Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.
4. The names “Xerces” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache\@apache.org.
5. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED “AS IS” AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Trademarks

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
CICS
DB2
IBM
IMS
IMS/ESA
iSeries
Language Environment
MQSeries
MVS
OS/390
OS/400
SecureWay
SupportPac
VisualAge
VSE/ESA
WebSphere
z/OS

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are registered 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.

Other company, product, or service names may be the trademarks or service marks of others.

Index

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-6065-00