

Application Messaging Interface

Note!				

Sixth edition (December 2000)

This edition applies to $IBM^{\textcircled{m}}$ MQSeries Application Messaging Interface Version 1.1, and to any subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1999, 2000. All rights reserved. US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract

with IBM Corp.

Contents

Figures xv	Initial values for structures	. 29
	C applications on AIX	. 30
Tables xvii	C applications on AS/400	. 31
Tubles	C applications on HP-UX	. 32
Alegant their hearts	C applications on OS/390	. 33
About this book xix	C applications on Solaris	
Who this book is for xix	C applications on Windows	
What you need to know to understand this book xix	11	
Structure of this book xix	Chapter 3. The C high-level interface	37
Appearance of text in this book xx	Overview of the C high-level interface	
	Initialize and terminate	
Summary of changes xxi	Sending messages	
Changes for this edition (SC34-5604-05) xxi	Receiving messages	
Changes for the fifth edition (SC34-5604-04) xxii	File transfer	
Changes for the fourth edition (SC34-5604-03) xxii	Publish/subscribe	
Changes for the third edition (SC34-5604-02) xxii	Transaction support	
	Reference information for the C high-level interface	39
Part 1. Introduction 1	amBackout	
Part I. Introduction	amBegin	
	amBrowseMsg	
Chapter 1. Introduction 3	amCommit	
Main components of the AMI	amInitialize	
Sending and receiving messages		
Interoperability	amPublish	
Programming languages 4	amReceiveFile	
Description of the AMI 4	amReceiveMsg	
Messages 4	amReceivePublication	
Services	amReceiveRequest	. 53
Policies 6	amSendFile	
Application Messaging Interface model 7	amSendMsg	
Further information 8	amSendRequest	
	amSendResponse	
Dort O. The C interfere	amSubscribe	
Part 2. The C interface 9	amTerminate	
	amUnsubscribe	. 61
Chapter 2. Using the Application		
Messaging Interface in C 13	Chapter 4. C object interface overview	63
Structure of the AMI		. 64
Using the repository		. 64
System default objects	Create objects	. 64
Writing applications in C	Get object handles	. 64
Opening and closing a session 16	Delete objects	. 65
Sending messages	Transactional processing	
Receiving messages	Error handling	
Request/response messaging	Message interface functions	. 66
File transfer	Get values	. 66
Publish/subscribe messaging	Set values	
Using name/value elements	Reset values	
Error handling	Read and write data	
Transaction support	Publish/subscribe topics	
	Publish/subscribe filters	
Sending group messages	Publish/subscribe name/value elements	
Other considerations	Error handling	
Using the AMI OAMAS subset	Publish/subscribe helper macros	
Building C applications	Sender interface functions	. 68
AMI include file	Open and close	68
Data types 29	open and crose	. 50

Send	. 68	amSesOpen	
Get values		amSesRollback	89
Error handling	. 68	Message interface functions	90
Receiver interface functions	. 69	amMsgAddElement	91
Open and close	. 69	amMsgAddFilter	91
Receive and browse	. 69	amMsgAddTopic	92
Get values	. 69	amMsgClearErrorCodes	
Set values		amMsgDeleteElement	
Error handling	. 69	amMsgDeleteFilter	93
Distribution list interface functions		amMsgDeleteNamedElement	93
Open and close		amMsgDeleteTopic	
Send		amMsgGetCCSID	
Get values		amMsgGetCorrelId	
Error handling		amMsgGetDataLength	95
Publisher interface functions		amMsgGetDataOffset	95
Open and close		amMsgGetElement	
Publish		amMsgGetElementCCSID	96
Get values		amMsgGetElementCount	96
Error handling		amMsgGetEncoding	96
Subscriber interface functions	72	amMcaCotFiltor	90
Open and close		amMsgGetFilter	97
		amMsgGetFilterCount	00
Broker messages			
Get values		amMsgGetGroupStatus	
Set value		amMsgGetLastError	
Error handling		amMsgGetMsgId	99
Policy interface functions	. 73	amMsgGetName	100
Get values		amMsgGetNamedElement	100
Set value		amMsgGetNamedElementCount	
Error handling		amMsgGetReportCode	
High-level functions	. 74	amMsgGetTopic	
		amMsgGetType	102
Chapter 5. C object interface reference	77	amMsgGetType	103
		amMsgGetType	103 103
Session interface functions	. 78	amMsgGetType	103 103 103
Session interface functions	. 78 . 78	amMsgGetType.	103 103 103 104
Session interface functions	. 78 . 78 . 78	amMsgGetType	103 103 103 104 104
Session interface functions	. 78 . 78 . 78 . 79	amMsgGetType.	103 103 103 104 104
Session interface functions amSesBegin	. 78 . 78 . 78 . 79 . 79	amMsgGetType	103 103 103 104 104 104
Session interface functions amSesBegin	. 78 . 78 . 78 . 79 . 79	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID	103 103 104 104 104 105
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreate amSesCreateDistList	. 78 . 78 . 78 . 79 . 79 . 79 . 80	amMsgGetType	103 103 104 104 104 105 105
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreate amSesCreateDistList amSesCreateMessage	. 78 . 78 . 78 . 79 . 79 . 79 . 80 . 80	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID	103 103 104 104 104 105 105
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy	. 78 . 78 . 78 . 79 . 79 . 79 . 80 . 80	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetEncoding amMsgSetFormat amMsgSetGroupStatus	103 103 104 104 104 105 105 106
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher	. 78 . 78 . 78 . 79 . 79 . 79 . 80 . 80 . 80 . 81	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetEncoding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes	103 103 103 104 104 105 105 106 106
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateReceiver	. 78 . 78 . 78 . 79 . 79 . 79 . 80 . 80 . 80 . 81	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros	103 103 104 104 104 105 105 106 106 107
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateReceiver amSesCreateSender	. 78 . 78 . 78 . 79 . 79 . 79 . 80 . 80 . 80 . 81 . 81	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElencoding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName	103 103 104 104 104 105 105 106 107 108
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateReceiver amSesCreateSender amSesCreateSubscriber	. 78 . 78 . 78 . 79 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp	103 103 103 104 104 1105 1105 1106 1107 1108 1108
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateReceiver amSesCreateSender amSesCreateSubscriber amSesDelete	. 78 . 78 . 78 . 79 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82	amMsgGetType. amMsgGetTopicCount amMsgReadBytes. amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset. amMsgSetElementCCSID amMsgSetElcoding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes. Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName.	103 103 104 104 105 105 106 107 108 108 108
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateReceiver amSesCreateSender amSesCreateSubscriber amSesDelete amSesDelete amSesDeleteDistList	. 78 . 78 . 78 . 79 . 79 . 79 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElcoding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName Sender interface functions	103 103 104 104 104 105 105 106 107 108 108 108 1109
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreateReceiver amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteMessage	. 78 . 78 . 78 . 79 . 79 . 79 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetFormat amMsgSetForopStatus amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName. Sender interface functions amSndClearErrorCodes	103 103 104 104 104 105 105 106 107 108 108 108 1109 1110
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreateReceiver amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteMessage amSesDeleteMessage	. 78 . 78 . 78 . 79 . 79 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 83	amMsgGetType. amMsgGetTopicCount amMsgReadBytes. amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetFormat amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName. Sender interface functions amSndClearErrorCodes amSndClose.	103 103 104 104 104 105 106 107 108 108 109 110 110
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreateReceiver amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePublisher	. 78 . 78 . 78 . 79 . 79 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 83	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetFormat amMsgSetForopStatus amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName. Sender interface functions amSndClearErrorCodes amSndClose amSndGetCCSID	103 103 104 104 104 105 106 107 108 108 109 110 110 111
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreateReceiver amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteMessage amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeleteReceiver	. 78 . 78 . 78 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 83 . 84 . 84	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetFormat amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName Sender interface functions amSndClearErrorCodes amSndClose amSndGetCCSID amSndGetCCSID amSndGetEncoding	103 103 104 104 104 105 105 106 107 108 108 109 110 110 111
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePolicy amSesDeleteMessage	. 78 . 78 . 78 . 79 . 79 . 79 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 83 . 84 . 84	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetFormat amMsgSetForupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName. Sender interface functions amSndClearErrorCodes amSndClose amSndGetCCSID amSndGetEncoding amSndGetEncoding amSndGetEncoding amSndGetLastError	103 103 104 104 104 105 105 106 107 108 108 110 110 111 111 111
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreateReceiver amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender amSesDeleteSender amSesDeleteSubscriber	. 78 . 78 . 78 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetErooding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName. Sender interface functions amSndClearErrorCodes amSndClose amSndGetCCSID amSndGetEncoding amSndGetEncoding amSndGetEncoding amSndGetLastError amSndGetName	103 103 104 104 104 105 105 106 107 108 108 109 110 111 111 111 111
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePolicy amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle	. 78 . 78 . 78 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85	amMsgGetType. amMsgGetTopicCount amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetElementCCSID amMsgSetFormat amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName. Sender interface functions amSndClearErrorCodes amSndClose amSndClose amSndGetCCSID amSndGetEncoding amSndGetLastError amSndGetName amSndGetName amSndOpen.	103 103 104 104 104 105 105 106 107 108 108 110 110 111 111 111 111 111
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePolicy amSesDeleteNessage amSesDeletePolicy amSesDeleteNessage amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender amSesDeleteSender amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle amSesGetLastError	. 78 . 78 . 78 . 78 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 86	amMsgGetType. amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelld. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetEncoding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes. Message interface helper macros AmMsgGetPubTimeStamp AmMsgGetStreamName Sender interface functions amSndClearErrorCodes amSndClose amSndGetCCSID amSndGetEncoding amSndGetLastError amSndGetName amSndOpen amSndOpen amSndOpen amSndSend	103 103 104 104 104 105 105 106 107 108 108 109 110 110 111 111 111 111 111 111
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePolicy amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle	. 78 . 78 . 78 . 78 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 86	amMsgGetType. amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetEncoding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes. Message interface helper macros AmMsgGetPubTimeStamp AmMsgGetStreamName Sender interface functions amSndClearErrorCodes amSndClose amSndGetCCSID amSndGetLastError amSndGetName amSndOpen amSndSend amSndSend amSndSend amSndSendFile.	103 103 104 104 104 105 105 106 107 108 108 109 110 111 111 111 111 111 111 111 111
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePublisher amSesDeleteDistList amSesDeleteNessage amSesDeletePolicy amSesDeletePublisher amSesDeletePublisher amSesDeletePolicy amSesDeletePublisher amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle amSesGetLastError amSesGetMessageHandle. amSesGetMessageHandle	. 78 . 78 . 78 . 78 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 84 . 84 . 85 . 85 . 85 . 86 . 86	amMsgGetType. amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetEncoding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes. Message interface helper macros AmMsgGetPubTimeStamp AmMsgGetStreamName Sender interface functions amSndClearErrorCodes amSndGetCCSID amSndGetLastError amSndGetName amSndGetName amSndGend amSndSend amSndSend amSndSend amSndSendFile Receiver interface functions.	103 103 104 104 104 105 105 106 107 108 108 109 110 111 111 111 111 111 111 111 111
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePublisher amSesDeleteNessage amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePublisher amSesDeletePublisher amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle amSesGetLastError amSesGetMessageHandle amSesGetPolicyHandle amSesGetPublisherHandle	. 78 . 78 . 78 . 78 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 85 . 86 . 87 . 87	amMsgGetType. amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetEncoding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes. Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName. Sender interface functions amSndClearErrorCodes amSndClose amSndGetCCSID amSndGetEncoding amSndGetLastError amSndGetName amSndOpen amSndSend amSndSend amSndSendFile Receiver interface functions amRcvBrowse	103 103 104 104 105 105 106 107 108 108 110 110 111 111 111 111 111 111
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePublisher amSesDeleteDistList amSesDeleteNessage amSesDeletePolicy amSesDeletePublisher amSesDeletePublisher amSesDeletePolicy amSesDeletePublisher amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle amSesGetLastError amSesGetMessageHandle. amSesGetMessageHandle	. 78 . 78 . 78 . 78 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 85 . 86 . 87 . 87	amMsgGetType. amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCOrrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetEncoding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName. Sender interface functions amSndClearErrorCodes amSndGetCCSID amSndGetLastError amSndGetLastError amSndGetName amSndOpen amSndSend amSndSend amSndSendFile Receiver interface functions amRcvBrowse amRcvBrowseSelect	103 103 104 104 105 105 106 107 108 108 108 110 111 111 111 111 111 111
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePublisher amSesDeleteNessage amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePublisher amSesDeletePublisher amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle amSesGetLastError amSesGetMessageHandle amSesGetPolicyHandle amSesGetPublisherHandle	. 78 . 78 . 78 . 78 . 79 . 79 . 80 . 80 . 80 . 81 . 81 . 82 . 82 . 83 . 83 . 83 . 83 . 83 . 84 . 84 . 85 . 85 . 86 . 86 . 87 . 87	amMsgGetType. amMsgReadBytes amMsgReset. amMsgSetCCSID amMsgSetCorrelId. amMsgSetDataOffset amMsgSetElementCCSID amMsgSetEncoding amMsgSetFormat amMsgSetGroupStatus amMsgWriteBytes. Message interface helper macros AmMsgAddStreamName AmMsgGetPubTimeStamp AmMsgGetStreamName. Sender interface functions amSndClearErrorCodes amSndClose amSndGetCCSID amSndGetEncoding amSndGetLastError amSndGetName amSndOpen amSndSend amSndSend amSndSendFile Receiver interface functions amRcvBrowse	103 103 104 104 105 105 106 107 108 108 108 119 110 111 111 111 111 111 111 115 115 117

amRcvGetDefnType	Deleting objects	. 152
amRcvGetLastError	Sending messages	
amRcvGetName	Receiving messages	. 153
amRcvGetQueueName	Request/response messaging	. 154
amRcvOpen	File transfer	. 155
amRcvReceive	Publish/subscribe messaging	. 156
amRcvReceiveFile	Using AmElement objects	. 157
amRcvSetQueueName	Error handling	
Distribution list interface functions	Transaction support	
amDstClearErrorCodes	Sending group messages	
amDstClose	Other considerations	. 159
amDstGetLastError	Building C++ applications	
amDstGetName	AMI include files	. 161
amDstGetSenderCount	C++ applications on AIX	. 161
amDstGetSenderHandle	C++ applications on AS/400	
amDstOpen	C++ applications on HP-UX	
amDstSend	C++ applications on Solaris	
amDstSendFile	C++ applications on Windows	
Publisher interface functions		
amPubClearErrorCodes	Chapter 7. C++ interface overview .	167
amPubClose	Base classes	
amPubGetCCSID	Helper classes	167
amPubGetEncoding	Exception classes	
amPubGetLastError	AmSessionFactory	
amPubGetName	Constructor	
amPubOpen	Session factory management	
amPubPublish	Create and delete session	
Subscriber interface functions	AmSession	
amSubClearErrorCodes	Session management	
amSubClose	Create objects	
amSubGetCCSID		
amSubGetDefnType	Delete objects	160
amSubGetEncoding	Error handling	170
amSubGetLastError	AmMessage	
amSubGetName	Get values	
amSubGetQueueName	Set values	
amSubOpen	Reset values	
amSubReceive	Read and write data	
amSubSetQueueName	Publish/subscribe topics	
amSubSubscribe	Publish/subscribe filters	
amSubUnsubscribe	Publish/subscribe name/value elements	
Policy interface functions	Error handling	
amPolClearErrorCodes	AmSender	
amPolGetLastError	Open and close	
amPolGetName	Send	
amPolGetWaitTime	Send file	
amPolSetWaitTime	Get values	
Part 3. The C++ interface 145	Error handling	
Part 3. The C++ interface 145		
	Open and close	
Chapter 6. Using the Application	Receive and browse	
Messaging Interface in C++ 149	Receive file	
Structure of the AMI	Get values	
Base classes	Set value	
Interface and helper classes	Error handling	
Exception classes	AmDistributionList	
Using the repository	Open and close	
System default objects	Send	
Writing applications in C++	Send file	
Creating and opening objects	Get values	
creating and opening objects	Error handling	. 175

AmPublisher	176	enableWarnings
Open and close		getLastErrorStatus
Publish		getName
Get values		getTraceLevel
Error handling		getTraceLocation
AmSubscriber		open
Open and close		rollback
Broker messages		AmMessage
Get values		addElement
Set value		addFilter
Error handling		addTopic
AmPolicy		clearErrorCodes
Policy management		deleteElement
Error handling		deleteFilter
Helper classes		deleteNamedElement
AmBytes		deleteTopic
AmElement		enableWarnings
AmObject		getCCSID
AmStatus		getCorrelationId
AmString	180	getDataLength
Exception classes	181	getDataOffset
AmException	181	getElement
AmErrorException	181	getElementCCSID
AmWarningException	181	getElementCount
0 1		getEncoding
Chapter 8. C++ interface reference	183	getFilter
Base classes		getFilterCount
		getFormat
Helper classes		getGroupStatus
Exception classes		getLastErrorStatus
AmSessionFactory		getMessageId
AmSessionFactory		getName
createSession		getNamedElement
deleteSession		getNamedElementCount
getFactoryName		
getLocalHost		getReportCode
getRepository		getTopic
getTraceLevel		getTopicCount
getTraceLocation		getType
setLocalHost	185	readBytes
setRepository	185	reset
setTraceLevel	185	setCCSID
setTraceLocation		setCorrelationId
AmSession		setDataOffset
begin		setElementCCSID
clearErrorCodes		setEncoding
close		setFormat
commit		setGroupStatus
createDistributionList		writeBytes
createMessage		AmSender
createPolicy		clearErrorCodes
createPublisher		close
		enableWarnings
createReceiver		getCCSID
createSender		getEncoding
createSubscriber		getLastErrorStatus
deleteDistributionList		
deleteMessage		getName
deletePolicy		open
deletePublisher		send
deleteReceiver		sendFile
deleteSender		AmReceiver
deleteSubscriber	188	browse

clearErrorCodes	operators	
close	pad	. 214
enableWarnings 202	AmElement	
getDefinitionType 202	AmElement	. 215
getLastErrorStatus 203	getName	. 215
getName	getValue	. 215
getQueueName 203	getVersion	. 215
open	setVersion	. 215
receive	toString	. 215
receiveFile	AmObject	
setQueueName	clearErrorCodes	. 216
AmDistributionList 205	getLastErrorStatus	. 216
clearErrorCodes 205	getName	
close	AmStatus	
enableWarnings 205	AmStatus	
getLastErrorStatus 205	getCompletionCode	
getName	getReasonCode	
getSender 205	getReasonCode2	
getSenderCount 205	toString	
open	AmString	
send	cat	
sendFile	cmp	
AmPublisher	constructors	
clearErrorCodes	contains	
close	сру	
enableWarnings	destructor	
getCCSID	operators	
getEncoding	pad	
getLastErrorStatus	split	
getName	strip	
open	length	
publish	text	
AmSubscriber	truncate	
clearErrorCodes	AmException	
close	getClassName	
enableWarnings	getCompletionCode	
getCCSID	getMethodName	
getDefinitionType	getReasonCode	
getEncoding	getSource	
getLastErrorStatus	toString	
	AmErrorException.	
getName		
getQueueName	getClassName	
open	getCompletionCode	
receive	getMethodName	
setQueueName	getReasonCode	
subscribe	getSource	
	toString	
AmPolicy	AmWarningException	
clearErrorCodes	getClassName	
enableWarnings	getCompletionCode	
getLastErrorStatus	getMethodName	
getName	getReasonCode	
getWaitTime	getSource	
setWaitTime	toString	. 222
AmBytes		
cmp	Part 4. The COBOL interface	223
constructors		
сру	Chapter 9. Using the Application	
dataPtr		00=
destructor	Messaging Interface in COBOL	
length	Structure of the AMI	. 227

Using the repository	228	Publish/subscribe topics	. 273
System default objects		Publish/subscribe filters	. 273
Writing applications in COBOL	230	Publish/subscribe name/value elements	. 273
Opening and closing a session	230	Error handling	
Sending messages		Sender interface functions	
Receiving messages		Open and close	
Request/response messaging	234	Send	
File transfer	235	Get values	
Publish/subscribe messaging		Error handling	
Using name/value elements		Receiver interface functions	
Error handling		Open and close	
Transaction support		Receive and browse	
Sending group messages		Get values	
Other considerations	240	Set values	
Building COBOL applications		Error handling	
COBOL applications on OS/390		Distribution list interface functions	
		Open and close	
Chapter 10. The COBOL high-level		Send	
interface	//3	Get values	
		Error handling	
Overview of the COBOL high-level interface		Publisher interface functions	. 277
		Open and close	. 277
Sending messages		Publish	
Receiving messages		Get values	
File transfer		Error handling	
Publish/subscribe		Subscriber interface functions	. 278
Reference information for the COBOL high-level	2 44	Open and close	. 278
interface	246	Broker messages	
		Get values	. 278
AMHBECIN (backout)		Set value	. 278
AMHBEGIN (begin)		Error handling	. 278
AMHCMIT (commit)	2 4 9 251	Policy interface functions	. 279
AMHINIT (initialize)		Get values	. 279
AMHPB (publish)		Set value	. 279
AMHRCFL (receive file)	25/	Error handling	
AMHRCMS (receive message)		High-level functions	. 280
AMHRCPB (receive publication)			
AMHRCRQ (receive request)		Chapter 12. COBOL object interface	
AMHSNFL (send file)		reference	283
AMHSNMS (send message)		Session interface functions	
AMHSNRQ (send request)	264	AMSEBG (begin)	
AMHSNRS (send response)		AMSECLEC (clear error codes)	
AMHSB (subscribe)		AMSECL (close)	
AMHTERM (terminate)		AMSECM (commit)	
AMHUN (unsubscribe)		AMSECR (create)	
		AMSECRDL (create distribution list)	
Chapter 11. COBOL object interface		AMSECRMS (create message)	
	000	AMSECRPO (create policy)	
overview		AMSECRPB (create publisher)	
Session interface functions		AMSECRRC (create receiver)	
Session management		AMSECRSN (create sender)	
Create objects		AMSECRSB (create subscriber)	
Get object handles	270	AMSEDL (delete)	
Delete objects		AMSEDLDL (delete distribution list)	
Transactional processing.		AMSEDLMS (delete message)	
Error handling		AMSEDLPO (delete policy)	
Message interface functions		AMSEDLPB (delete publisher)	
Get values		AMSEDLRC (delete receiver)	
Set values		AMSEDLSN (delete sender)	
Reset values		AMSEDLSB (delete subscriber)	
Read and write data	L/ L	AMSEGHDL (get distribution list handle)	

AMSEGTLE (get last error codes)	293	AMRCBRSE (browse selection message) .	324
AMSEGHMS (get message handle)		AMRCCLEC (clear error codes)	
AMSEGHPO (get policy handle)		AMRCCL (close)	
AMSEGHPB (get publisher handle)		AMRCGTDT (get definition type)	
AMSEGHRC (get receiver handle)		AMRCGTLE (get last error)	
AMSEGHSN (get sender handle)		AMRCGTNA (get name)	
AMSEGHSB (get subscriber handle)		AMRCGTQN (get queue name)	328
AMSEOP (open)		AMRCOP (open)	
AMSERB (rollback)		AMRCRC (receive)	
Message interface functions		AMRCRCFL (receive file)	
AMMSADEL (add element)		AMRCSTQN (set queue name)	
AMMSADFI (add filter)		Distribution list interface functions	
AMMSADTO (add topic)			
		AMDLCLEC (clear error codes)	
AMMSCLEC (clear error codes)		AMDLCL (close)	
AMMSDEEL (delete element)		AMDLGTLE (get last error)	
AMMSDEFI (delete filter)		AMDLGTNA (get name)	333
AMMSDENE (delete named element)		AMDLGTSC (get sender count)	
AMMSDETO (delete topic)		AMDLGTSH (get sender handle)	
AMMSGELC (get element CCSID)		AMDLOP (open)	
AMMSGTCC (get CCSID)	301	AMDLSN (send)	335
AMMSGTCI (get correl ID)		AMDLSNFL (send file)	335
AMMSGTDL (get data length)		Usage notes	
AMMSGTDO (get data offset)		Publisher interface functions	
AMMSGTEL (get element)		AMPBCLEC (clear error codes)	
AMMSGTEC (get element count)		AMPBCL (close)	
AMMSGTEN (get encoding)	304	AMPBGTCC (get CCSID)	337
AMMSGTFC (get filter count)	304	AMPBGTEN (get encoding)	338
AMMSGTFI (get filter)	305	AMPBGTLE (get last error)	338
AMMSGTFO (get format)	305	AMPBGTNA (get name)	339
AMMSGTGS (get group status)		AMPBOP (open)	
AMMSGTLE (get last error)		AMPBPB (publish)	
AMMSGTMI (get message ID)		Subscriber interface functions	341
AMMSGTNA (get name)		AMSBCLEC (clear error codes)	
AMMSGTNE (get named element)	308	AMSBCL (close)	
AMMSGTNC (get named element count)		AMSBGTCC (get CCSID)	
AMMSGTRC (get report code)		AMSBGTDT (get definition type)	
AMMSGTTO (get topic)		AMSBGTEN (get encoding)	343
AMMSGTTC (get topic count)		AMSBGTLE (get last error)	
AMMSGTTY (get type)		AMSBGTNA (get name)	
AMMSREBY (read bytes)		AMSBGTQN (get queue name)	344
AMMSRS (reset)		AMSBOP (open)	2/5
AMMSSTCC (set CCSID)	312	AMSBRC (receive)	
AMMSSTCI (set correl ID)			
		AMSBSTQN (set queue name)	246
AMMSSTDO (set data offset)			
		AMSBUN (unsubscribe)	
AMMSSTEN (set encoding)		Policy interface functions	
AMMSSTFO (set format)		AMPOCILE (clear error codes)	
AMMSSTGS (set group status)		AMPOGTLE (get last error)	
AMMSWRBY (write bytes)		AMPOGTNA (get name)	
Sender interface functions		AMPOGTWT (get wait time)	
AMSNCLEC (clear error codes)		AMPOSTWT (set wait time)	350
AMSNCL (close)			
AMSNGTCC (get CCSID)		Part 5. The Java interface	. 351
AMSNGTEN (get encoding)			. 551
AMSNGTLE (get last error)	318	Chapter 12 Hainer the Ameliantian	
AMSNGTNA (get name)	318	Chapter 13. Using the Application	_
AMSNOP (open)		Messaging Interface in Java	
AMSNSN (send)		Structure of the AMI	355
AMSNSNFL (send file)		Base classes	355
Receiver interface functions		Interface and helper classes	
AMRCBR (browse)		Exception classes	

Using the repository	. 356	Publish	. 377
System default objects	. 356	Get values	. 377
Writing applications in Java	. 357	Error handling	. 377
Creating and opening objects	. 357	AmSubscriber	
Sending messages		Open and close	
Receiving messages		Broker messages	
Request/response messaging		Get values	
File transfer	. 361	Set value	
Publish/subscribe messaging		Error handling	
Using AmElement objects		AmPolicy	
Error handling		Policy management	
Transaction support		Error handling	
Sending group messages		Helper classes	
Other considerations		AmConstants	
Building Java applications		AmElement	
AMI package for Java		AmObject	
Running Java programs	. 300	AmStatus	
		Exception classes	. 201
Chapter 14. Java interface overview		AmException	
Base classes		AmErrorException	
Helper classes	. 369	AmWarningException	. 381
Exception classes	. 369		
AmSessionFactory	. 370	Chapter 15. Java interface reference	383
Constructor	. 370	Base classes	. 383
Session factory management	. 370	Helper classes	. 383
Create session		Exception classes	
AmSession		AmSessionFactory	
Session management		AmSessionFactory	. 384
Create objects		createSession	
Transactional processing		getFactoryName	
Error handling		getLocalHost	
AmMessage	372	getRepository	
Get values		getTraceLevel	384
Set values		getTraceLocation	. 304 384
Reset values		setLocalHost	
Read and write data		setRepository	
Publish/subscribe filters		setTraceLevel	
Publish/subscribe topics		setTraceLocation	
Publish/subscribe name/value elements		AmSession	
Error handling		begin	
AmSender			. 386
Open and close			. 386
Send			. 386
Send file			. 386
Get values	. 374	O	. 387
Error handling	. 374		. 387
AmReceiver	. 375	createPublisher	. 387
Open and close	. 375	createReceiver	. 387
Receive and browse	. 375	createSender	. 387
Receive file	. 375	createSubscriber	. 387
Get values	. 375	enableWarnings	. 388
Set value	. 375	· · · · · · · · · · · · · · · · · · ·	. 388
Error handling		0	. 388
AmDistributionList		8-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1	. 388
Open and close	0=1	6	. 388
Send	0=1	e e e e e e e e e e e e e e e e e e e	. 388
Send file	. 376	1	. 388
Get values			. 389 . 389
Error handling		addElement	
AmPublisher		addFilter	
Open and close		addTopic	
		aud 10ptc	. シラひ

clearErrorCodes	
deleteElement	0
deleteFilter	getLastErrorStatus
deleteNamedElement	getName
deleteTopic	391 getSender
enableWarnings	getSenderCount
getCCSID	open
getCorrelationId	
getDataLength	
getDataOffset	
getElement	391 clearErrorCodes
getElementCount	
getEncoding	
getFilter	
getFilterCount	
getFormat	ě ě
getGroupStatus	O .
getLastErrorStatus	
getMessageId	*
getName	
getNamedElement	
getNamedElementCount	
getReportCode	
getTopic	· · · · · · · · · · · · · · · · · · ·
getTopicCount	•
	getEncoding
	getLastErrorStatus
,	394 getName
setCCSID	0
setCorrelationId	
setDataOffset	*
	395 setQueueName
O	395 subscribe
setGroupStatus	
writeBytes	
,	396 clearErrorCodes
	enableWarnings
	getLastErrorStatus
enableWarnings	
getCCSID.	
getEncoding	
getLastErrorStatus	
INT.	
	397 AmElement
send	
sendFile	0
AmReceiver	
browse	9
clearErrorCodes	
close	
enableWarnings	
getDefinitionType	
getLastErrorStatus	
getName	
getQueueName	
open	
· .	
. 711	0
setQueueName	
1 7 6 1	
clearErrorCodes	403 getClassName

getCompletionCode 415	Setting the runtime environment
getMethodName 415	Directory structure (Solaris)
getReasonCode	Installation on Windows
getSource	Installation
toString	Setting the runtime environment
AmErrorException	Directory structure (Windows)
getClassName	Local host and repository files (AS/400, UNIX, and
getCompletionCode	Windows)
getMethodName	Default location
getReasonCode	Default names
getSource	Overriding the default location and names 450
toString	Local host file
AmWarningException	Repository file
getCompletionCode	Batch, RRS-batch, IMS
getMethodName	CICS
getReasonCode	Local host file
getSource	Repository file
toString	Repository and local host caches 454
tooting	The administration tool
Dowl C OC/000 Cubounters 440	Installation
Part 6. OS/390 Subsystems 419	Operation
	Connecting to MQSeries
Chapter 16. Writing applications for	Using MQSeries Integrator Version 1
OS/390 subsystems 421	Using MQSeries Publish/Subscribe
Writing IMS applications using AMI 421	Using MQSeries Integrator Version 2
Writing CICS applications using AMI 421	Migrating to MQSeries Integrator V2 from V1
Writing batch applications using AMI 422	and MQSeries Publish/Subscribe 459
Writing RRS-batch applications using AMI 422	Creating default MQSeries objects
RRS availability 422	The sample programs
, and the same of	Sample programs for AS/400, UNIX, and
Part 7 Sotting up an AMI	Windows
Part 7. Setting up an AMI	Running the AS/400, UNIX, and Windows
installation 423	sample programs
	Sample programs for OS/390 464
Chapter 17. Installation and sample	Running the sample programs (OS/390) 465
programs 425	
Prerequisites	Chapter 18. Defining services and
Disk space	policies 469
Operating environments 425	Services and policies
MQSeries environment 426	System provided definitions
Language compilers 426	System default objects 470
Installation on AIX 428	Service definitions
Installation	Service point (sender/receiver) 472
Setting the runtime environment 429	Distribution list 474
Directory structure (AIX) 430	Subscriber
Installation on AS/400	Publisher
Setting the runtime environment for Java	Policy definitions
programs	Initialization attributes 475
Directory structure (AS/400) 433	General attributes
Installation on HP-UX 436	Send attributes
Installation	Receive attributes
Setting the runtime environment 437	Subscribe attributes
Directory structure (HP-UX) 438	Publish attributes
Installation on OS/390	
Installation	Chapter 19. Problem determination 481
Setting the runtime environment 440	Using trace (AS/400, UNIX, and Windows) 48:
Unicode character conversion 440	Trace filename and directory
Directory structure (OS/390) 441	C++ and Java
Installation on Sun Solaris 443	Example trace
Installation	Example trace

Using trace (OS/390)	AMH (Handle constants) 510 AMLEN (String length constants) 510 AMMCD (Message Content Descriptor tag
GTF Trace	names)
Common causes of problems 493	Appendix C. Notices
Part 8. Appendixes 495 Appendix A. Reason codes 497	Glossary of terms and abbreviations 519
Reason code: OK	Bibliography
Reason code: Failed	MQSeries platform-specific publications
Appendix B. Constants	Portable Document Format (PDF)
AMFMT (Format constants) 510 AMGF and AMGRP (Group status constants) 510	Sending your comments to IBM 537

				_	_
-1	a	u	r	_	c
	9	u	•	_	J

Tables

1.	System default objects	8.	System default objects			. 470
2.	Object interface calls used by the high-level	9.	Service point (sender/receiver).			. 472
	functions	10.	Distribution list			. 474
3.	System default objects	11.	Subscriber			. 474
	Object interface calls used by the high-level		Publisher			
	functions	13.	Initialization attributes			. 475
5.	The sample programs for AS/400, UNIX, and	14.	General attributes			. 476
	Windows platforms 461	15.	Send attributes			. 476
6.	The sample programs for OS/390 ('batch'	16.	Receive attributes			. 478
	includes RRS-batch) 464	17.	Subscribe attributes			. 479
7.	System provided definitions 470	18.	Publish attributes			. 480

About this book

This book describes how to use the MQSeries 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 MQSeries 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 MQSeries 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 MQSeries 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 MQSeries Message Queue Interface (MQI). This is described in the MQSeries Application Programming Guide and the MQSeries 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 MOSeries Publish/Subscribe.
 - MQSeries Integrator Version 1.1 Application Development Guide if you are going to use the AMI with MQSeries Integrator Version 1.1.
 - MQSeries Integrator Version 2.0 Programming Guide if you are going to use the AMI with MQSeries Integrator Version 2.0.
- 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 MQSeries, 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 145 describes how to use the AMI in C++ programs.
- "Part 4. The COBOL interface" on page 223 describes how to write AMI programs using the COBOL high-level and object interfaces.
- "Part 5. The Java interface" on page 351 describes how to use the AMI in Java programs.
- "Part 6. OS/390 Subsystems" on page 419 gives advice on writing AMI applications for OS/390 $^{\circ}$ subsystems.

About this book

• "Part 7. Setting up an AMI installation" on page 423 is for systems administrators who are setting up an Application Messaging Interface installation.

Appearance of text in this book

This book uses the following type styles:

Format The name of a parameter in an MQSeries call, a field in an MQSeries structure, or an attribute of an MQSeries 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

Summary of changes

This section describes changes in this edition of *MQSeries Application Messaging Interface*. Changes since the previous edition of the book are marked by vertical lines to the left of the changes.

Changes for this edition (SC34-5604-05)

The changes to this edition of the Application Messaging Interface are: Updates to describe new support for the AS/400[®] system. There are minor changes throughout this manual, and more significant changes in the following sections: - "C applications on AS/400" on page 31 - "C++ applications on AS/400" on page 162 "Building Java applications" on page 366 - "Installation on AS/400" on page 432 - "Local host and repository files (AS/400, UNIX, and Windows)" on page 450 - "Using trace (AS/400, UNIX, and Windows)" on page 481 Editorial changes to clarify the usage notes (and some syntax) for the following calls: - "amBrowseMsg" on page 42 (C) "amReceiveMsg" on page 49 (C) "amReceiveRequest" on page 53 (C) "amRcvBrowse" on page 115 (C) "amRcvBrowseSelect" on page 117 (C) "amRcvReceive" on page 122 (C) "AMHBRMS (browse message)" on page 249 (COBOL) "AMHRCMS (receive message)" on page 256 (COBOL) "AMHRCRQ (receive request)" on page 260 (COBOL) "AMRCBR (browse)" on page 322 (COBOL) "AMRCBRSE (browse selection message)" on page 324 (COBOL) - "AMRCRC (receive)" on page 329 (COBOL) • Editorial changes to clarify the syntax of the following calls: "amRcvReceiveFile" on page 124 (C) - "AmReceiver" on page 201 (C++) - "AmDistributionList" on page 205 (C++) - "AmReceiver" on page 399 (Java) • Editorial changes to clarify the following sections: - "Sample programs for AS/400, UNIX, and Windows" on page 461

"Service definitions" on page 472
"Send attributes" on page 476
"Receive attributes" on page 478
"Subscribe attributes" on page 479

Changes for the fifth edition (SC34-5604-04)

This is the first edition of the book available in hardcopy form and contains several editorial changes, mainly for clarification of the following calls:

- browse a message (see "amRcvBrowse" on page 115 for C and "AMRCBR (browse)" on page 322 for COBOL)
- browse a selection message (see "amRcvBrowseSelect" on page 117 for C and "AMRCBRSE (browse selection message)" on page 324 for COBOL)

Changes for the fourth edition (SC34-5604-03)

This edition was not published.

Changes for the third edition (SC34-5604-02)

- Application Messaging Interface now provides support for applications written in the C and COBOL programming languages, running on the OS/390 operating system. See:
 - "Part 4. The COBOL interface" on page 223 for a description of the COBOL high-level and object interfaces.
 - "Part 6. OS/390 Subsystems" on page 419 for information about writing AMI applications for OS/390 subsystems.
- · New calls and methods have been included for:
 - file transfer
 - content-based publish/subscribe
 - returning the message type
 - returning the feedback code from a report message

See the parts of the book describing each supported language for details.

- New high-level calls have been added for both C and COBOL to:
 - browse a message (see "amBrowseMsg" on page 42 for C and "AMHBRMS (browse message)" on page 249 for COBOL)
 - begin a unit of work (see "amBegin" on page 41 for C and "AMHBEGIN (begin)" on page 248 for COBOL)
- Support is provided for MQSeries Integrator Version 2.0. For details see "Using MQSeries Integrator Version 2" on page 458.
- There is now a subset of the AMI C interface that conforms to the Open Application Group Middleware Application Program Interface Specification (OAMAS). See "Using the AMI OAMAS subset" on page 28 for details.

Part 1. Introduction

Chapter 1. Introduction		. 3
Main components of the AMI		. 3
Sending and receiving messages		. 3
Interoperability		. 3
Programming languages		
Description of the AMI		. 4
Messages		
Services		
Point-to-point and publish/subscribe		. 5
Types of service		. 5
Policies		
Application Messaging Interface model .		
Further information		

Chapter 1. Introduction

The MQSeries 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 MQSeries *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 MQSeries 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 MQSeries Publish/Subscribe or MQSeries Integrator)

Main components of the AMI

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 MQSeries 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 MQSeries 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:

Message ID An identifier for the message. It is usually unique, and typically it

is generated by the message transport (MQSeries).

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 <code>MessageID</code>

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 an MQSeries 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 MQSeries 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,

Description of the AMI

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, please refer to the MQSeries 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).

Description of the AMI

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 shows the components of the Application Messaging Interface.

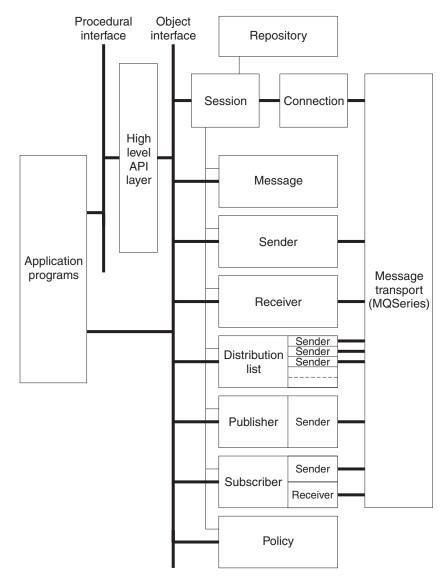


Figure 1. Basic AMI model

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 (MQSeries). 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

Application Messaging Interface model

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.

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 145
- "Part 4. The COBOL interface" on page 223
- "Part 5. The Java interface" on page 351

In "Part 6. OS/390 Subsystems" on page 419, you will find advice on writing AMI applications for the Information Management System (IMS), Customer Information Control System (CICS)[®], batch, and RRS-batch (recoverable resource services) subsystems on OS/390.

In "Part 7. Setting up an AMI installation" on page 423, 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

The Application Messaging Interface for C, C++, and Java runs on the following operating systems or environments: AIX[®], AS/400, HP-UX, Sun Solaris, Microsoft[®] Windows[®] 98 and Windows NT[®].

The Application Messaging Interface for C and COBOL runs on the OS/390 operating system.

Part 2. The C interface

Chapter 2. Using the Application Messaging			Chapter 3. The C high-level interface	. 37
Interface in C		. 13	Overview of the C high-level interface	. 38
Structure of the AMI		. 13	Initialize and terminate	. 38
Using the repository		. 14	Sending messages	. 38
System default objects		. 14	Receiving messages	
Writing applications in C		. 16	File transfer	. 38
Opening and closing a session		. 16	Publish/subscribe	
Sending messages		. 16	Transaction support	
Using the message object		. 17	Reference information for the C high-level interface	
Sample programs			amBackout	
Receiving messages			amBegin	
Using the message object		. 19	amBrowseMsg	
Sample programs			amCommit	. 44
Request/response messaging			amInitialize	
Request			amPublish	. 46
Response			amReceiveFile	. 47
Sample programs			amReceiveMsg	
File transfer		. 21	amReceivePublication	. 51
Publish/subscribe messaging			amReceiveRequest	
Publish			amSendFile	. 55
Subscribe			amSendMsg	
Sample programs			amSendRequest	
Using name/value elements			amSendResponse	
Parameters			amSubscribe	. 59
Example			amTerminate	
Error handling			amUnsubscribe	
Transaction support				
Sending group messages			Chapter 4. C object interface overview	63
Other considerations		. 27	Session interface functions	. 64
Multithreading			Session management	
Using MQSeries with the AMI		. 27	Create objects	. 64
Field limits			Create objects	. 64
Using the AMI OAMAS subset			Delete objects	. 6
Building C applications			Transactional processing	
AMI include file		. 29	Error handling	
Data types		. 29	Message interface functions	. 66
Initial values for structures		29	Get values	. 66
C applications on AIX	•	30	Set values	. 66
Preparing C programs on AIX			Reset values	
Running C programs on AIX	•	30	Read and write data	
C applications on AS/400			Publish/subscribe topics	
Preparing C programs on AS/400		31	Publish/subscribe filters	. 67
Running C programs on AS/400			Publish/subscribe name/value elements	
C applications on HP-UX			Error handling	
Preparing C programs on HP-UX			Publish/subscribe helper macros	. 67
Running C programs on HP-UX			Sender interface functions	
C applications on OS/390				
Preparing C programs on OS/390			Open and close	. 00
Running C programs on OS/390				
C applications on Solaris			Get values	
Preparing C programs on Solaris			Error handling	
Running C programs on Solaris			Receiver interface functions	
C applications on Windows			Open and close	
Preparing C programs on Windows			Receive and browse	
Running C programs on Windows			Get values	
Rumming C programs on windows		. 55	Set values	
			Error handling	. 05

Distribution list interface functions		amMsgDeleteFilter
Open and close	70	Parameters
Send		amMsgDeleteNamedElement
Get values		amMsgDeleteTopic 94
Error handling		amMsgGetCCSID 94
Publisher interface functions		amMsgGetCorrelId 94
Open and close		amMsgGetDataLength95
Publish		amMsgGetDataOffset 95
Get values		amMsgGetElement 95
Error handling		amMsgGetElementCCSID 96
Subscriber interface functions		amMsgGetElementCount 96
Open and close		amMsgGetEncoding 96
Broker messages		amMsgGetFilter
Get values	72	Parameters
Set value		amMsgGetFilterCount
Error handling		Parameters
Policy interface functions		amMsgGetFormat
Get values	73	amMsgGetGroupStatus
Set value		amMsgGetLastError
Error handling		amMsgGetMsgId
High-level functions	74	amMsgGetName
		amMsgGetNamedElement
Chapter 5. C object interface reference	77	amMsgGetNamedElementCount 101
Session interface functions		amMsgGetReportCode
amSesBegin	78	amMsgGetTopic
amSesClearErrorCodes		amMsgGetType
amSesClose		amMsgGetTopicCount
amSesCommit		amMsgReadBytes
amSesCreate	79	amMsgReset
amSesCreateDistList	80	amMsgSetCCSID
amSesCreateMessage		amMsgSetCorrelId
amSesCreatePolicy		amMsgSetDataOffset
amSesCreatePublisher	81	amMsgSetElementCCSID
amSesCreateReceiver	81	amMsgSetEncoding
amSesCreateSender		amMsgSetFormat
amSesCreateSubscriber		amMsgSetGroupStatus
amSesDelete		amMsgWriteBytes
amSesDeleteDistList		Message interface helper macros
amSesDeleteMessage		AmMsgAddStreamName
amSesDeletePolicy		AmMsgGetPubTimeStamp
amSesDeletePublisher		AmMsgGetStreamName
amSesDeleteReceiver		Sender interface functions
amSesDeleteSender		amSndClearErrorCodes
amSesDeleteSubscriber		amSndClose
amSesGetDistListHandle		amSndGetCCSID
amSesGetLastError		amSndGetEncoding
amSesGetMessageHandle		amSndGetLastError
amSesGetPolicyHandle		amSndGetName
amSesGetPublisherHandle		amSndOpen
amSesGetReceiverHandle		amSndSend
amSesGetSenderHandle		amSndSendFile
amSesGetSubscriberHandle		Parameters
amSesOpen		Usage notes
amSesRollback		Receiver interface functions
Message interface functions		amRcvBrowse
amMsgAddElement		Usage notes
amMsgAddFilter		amRcvBrowseSelect
Parameters		Usage notes
amMsgAddTopic		amRcvClearErrorCodes
amMsgClearErrorCodes		amRcvClose
amMsgDeleteElement		amRcvGetDefnType
anning Deferencement	14	uniter General ype

	amRcvGetLastErro												
	amRcvGetName												120
	amRcvGetQueueN	am	e										121
	amRcvOpen												121
	amRcvOpen amRcvReceive . Usage notes . amRcvReceiveFile												122
	Usage notes .												122
	amRcvReceiveFile												124
	Usage notes .												125
	amRcvSetOueueNa	m	e										125
Di	stribution list interf amDstClearErrorCo amDstClose amDstGetLastError	асє	e fu	ınct	tior	าร							126
	amDstClearErrorCo	ode	es										126
	amDstClose												126
	amDstGetLastErroi	•											127
	amDstGetName amDstGetSenderCo												127
	amDstGetSenderCo	our	nt										128
	amDstGetSenderH	ano	lle										128
	amDstOpen	A114	arc	•	•	•	•	•	•	•	•	•	128
	amDstSend	•	•	•	•	•	•	•	•	•	•	•	129
	amDstGetSenderHamDstOpen amDstSend amDstSendFile .	•	•	•	•	•	•	•	•	•	•	•	130
	Parameters .	•	•	•	•	•	•	•	•	•	•	•	130
	Heara notes	•	•	•	•	•	•	•	•	•	•	•	130
D11	blicher interface fu	act	ion		•	•	•	•	•	•	•	•	131
LU	Usage notes . blisher interface fur amPubClearErrorC amPubClose . amPubGetCCSID	od	2011	13	•	•	•	•	•	•	•	•	131
	amPubCloso	ou	.03	•	•	•	•	•	•	•	•	•	131
	amPubCotCCSID	•	•	•	•	•	•	•	•	•	•	•	131
	amPubGetEncoding	~	•	•	•	•	•	•	•	•	•	•	122
	and ubGetEncount	5	•	•	•	•	•	•	•	•	•	•	132
	amPubGetLastErro amPubGetName amPubOpen amPubPublish .	ľ	•	•	•	•	•	•	•	•	•	•	132
	amPubOn an	•	•	•	•	•	•	•	•	•	•	•	122
	amPubOpen	•	•	•	•	•	•	•	•	•	•	•	133
_	ampubpublish .	•		•	•	•	•	•	•	•	•	•	134
5u	bscriber interface fu	ınc	tio	ns	•	•	•	•	•	•	•	•	135
	amSubClearErrorC	oa	es	•	•	•		•	•	•	•	•	135
	amSubClose amSubGetCCSID amSubGetDefnTyp amSubGetEncoding	•	•	•	•	•	•	•	•	•	•	•	135
	amSubGetCCSID	•	•	•	•	•	•	•	•	•	•	•	135
	amSubGetDetnTyp	e	•	•								•	136
	amSubGetEncoding	5	•	•								•	136
	amSubGetLastErro	r	•	•		•		•		•	•	•	137
	amSubGetName amSubGetQueueN amSubOpen amSubReceive .												137
	amSubGetQueueN	am	ıe	•									138
	amSubOpen												138
	amSubReceive .												139
	amoubbeiQueuerva	ımı	e										139
	amSubSubscribe												140
	amSubUnsubscribe	:											141
Po	licy interface functi	on	s										142
	amPolClearErrorCo	ode	es										142
	amPolGetLastError												142
	amPolGetName.												143
	amPolGetWaitTime	•											143
	amPolSetWaitTime												143

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 MQSeries users.

This chapter describes the following:

- "Structure of the AMI"
- "Writing applications in C" on page 16
- "Building C applications" on page 29

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 MQSeries '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 an MQSeries

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 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 <code>MessageID</code>), or to change the value of one or more properties before sending the message (for example, the <code>Format</code>). 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 <code>set</code> and <code>get</code> their properties. You can mix high-level and object-level functions in the same application.

Structure of the AMI

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 37. An overview of the object interface is given in "Chapter 4. C object interface overview" on page 63, with reference information in "Chapter 5. C object interface reference" on page 77.

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 470).

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
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 470).

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).

Structure of the AMI

Constants representing synonyms for handles are also provided for these objects, for use from the object interface (see "Appendix B. Constants" on page 509). 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).

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 74).

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 45):

```
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 60):

```
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 56). 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 Message ID, 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 66 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 103) 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 AS/400, UNIX, and Windows" on page 461).

Receiving messages

Use the amReceiveMsg high-level function (page 49) 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

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 AS/400, UNIX, and Windows" on page 461).

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 57) 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

The amReceiveRequest high-level function (page 53) 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 58):

Sending a response message

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 AS/400, UNIX, and Windows" on page 461).

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.

To ensure that the file can be reassembled at the receiving side during end-to-end file transfer, you should use a policy with the 'physical splitting' file transfer option. 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.

```
Sending a file using the high-level amSendFile function
```

When using physical splitting, the AMI may send a group of messages rather than 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 26 for more information. Because errors may occur part way through sending or receiving a file, applications must ensure that the transfer completed as expected. In particular, we recommend that file transfers are done with the syncpoint policy option turned on, and that applications check the reason and completion codes carefully to be sure that the whole file was sent before committing the unit of work.

```
Receiving a file using the high-level amReceiveFile function
```

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, we recommend the use of the policy syncpoint option and checking of 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 OS/390 datasets with different record types or sizes, and between OS/390 datasets and the flat files used on other systems.

If the intent is not to transfer a file from one location to another, but rather to generate a group of messages from a file, you should 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 OS/390 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.

Note that file transfer calls are not supported under CICS. All of the calls (amSendFile, amReceiveFile, amSndSendFile, amRcvReceiveFile, and amDstSendFile) will return an error with reason code AMRC_FILE_TRANSFER_INVALID (144) if used in a CICS application running on OS/390.

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 MQSeries Integrator Version 2" on page 458).

For more information, refer to the MQSeries Integrator Version 2 Programming Guide or the MQSeries Publish/Subscribe User's Guide.

Publish

Use the amPublish high-level function (page 46) 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 59) 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, OL, 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 61). 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 51). 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 AS/400, UNIX, and Windows" on page 461).

Using name/value elements

Publish/subscribe brokers (such as MQSeries 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 {
                              /* Structure identifier
  AMCHAR8 strucId;
                        /* Structure version number */
  AMLONG version;
  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
            pName; /* Name
valueBuffLen; /* Value buffer length
  AMSTR
  AMLONG
           valueLen; /* Value length in bytes pValue; /* Value
  AMLONG
 AMLONG typeBuffLen; /* Reserved, must be zero
AMLONG typeLen; /* Reserved, must be zero
AMSTR pType: /* Posserved
            pType;
 } AMELEM;
```

See "Initial values for structures" on page 29 for advice on initialization of this structure.

Parameters

i didiliotolo	
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, OL, &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 MQSeries 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

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 depends on the type of transaction is being used.

- · MQSeries 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 MQSeries 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).
 - MQSeries cannot be used as an XA transaction coordinator on OS/390.
- 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
{\rm AMGRP\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\midddu\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middle\middd
AMGRP LAST MSG IN GROUP for the last message
```

The message status is set using amMsgSetGroupStatus.

Although native group message support is not available using MQSeries for OS/390 Version 5.2, group messages can be sent and received using AMI by selecting 'Simulated Group Support' in the repository service point definitions of the sender and receiver services used by the applications. Group messages are sent and received by an application in exactly the same way regardless of whether 'Simulated Group Support' is enabled for the repository service definitions.

Certain restrictions apply when 'Simulated Group Support' is enabled. These are as follows:

- Applications may not set or use the correlation id.
- A message that is not part of a group will be sent as a group of one message (that is, its group flags will be 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.

Note that if MQSeries for OS/390 Version 5.2 is involved in any way in sending or receiving group messages or files, 'Simulated Group Support' must be enabled on both the sending and receiving systems. This applies even if one of the systems is not an OS/390 platform.

Other considerations

You should consider the following when writing your applications:

- Multithreading
- Using MQSeries 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 (MQSeries) 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 MQSeries and a transport error (AMRC_TRANSPORT_ERR) will be reported to the application.

Multithreaded applications are not supported on OS/390.

Using MQSeries with the AMI

You must not mix MQSeries 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 MQSeries, the underlying message transport. See the rules for naming MQSeries objects in the MQSeries 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 http://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

I

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
                  (AS/400)
                  (0S/390)
hlq.SCSQC370
/amt/inc
                  (UNIX)
\amt\include
                  (Windows)
```

See "Directory structure" on page 430 (AIX), page 433 (AS/400), page 438 (HP-UX), page 441 (OS/390), page 445 (Solaris), or page 448 (Windows).

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.

Building C applications

Next step

Now go to one of the following to continue building a C application:

- "C applications on AIX"
- "C applications on AS/400" on page 31
- "C applications on HP-UX" on page 32
- "C applications on OS/390" on page 33
- "C applications on Solaris" on page 34
- "C applications on Windows" on page 35

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 "Language compilers" on page 426 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/mgm/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 -1 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 428).

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
In -s /usr/mqm/lib/libamtXML310.a /usr/lib/libamtXML310.a
In -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 MQSeries runtime binding stubs available in your runtime environment. These stubs allow AMI to load MQSeries libraries dynamically.

For the non-threaded MQSeries Server library, perform:

```
ln -s /usr/mgm/lib/amtcmgm /usr/lib/amtcmgm
```

For the non-threaded MQSeries Client library, perform:

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

For the threaded MQSeries Server library, perform:

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

For the threaded MQSeries Client library, perform:

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

C applications on AS/400

I

I

I

ı ı

1

I

ı

This section explains what you have to do to prepare and run your C programs on the AS/400 system. See "Language compilers" on page 426 for compilers supported by the AMI.

Preparing C programs on AS/400

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 AS/400

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 HP-UX

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 "Language compilers" on page 426 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 -W1,+b,:,-L flags. In the case of HP-UX, they are usually located at /opt/mgm/lib.

• Link with the AMI library. To do this, use the -1 flag, more specifically -lamt.

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

```
aCC +DAportable -W1,+b,:,-L/opt/mgm/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.

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 436).

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
In -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
In -s /opt/mqm/lib/libamtICUUC140 r.sl /usr/lib/libamtICUUC140 r.sl
```

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

For the non-threaded MQSeries Server library, perform:

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

For the non-threaded MQSeries Client library, perform:

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

For the threaded MQSeries Server library, perform:

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

For the threaded MQSeries 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 OS/390

This section explains what you have to do to prepare and run your C programs on the OS/390 operating system. See "Language compilers" on page 426 for compilers supported by the AMI.

Preparing C programs on OS/390

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 OS/390

The AMI needs access to the MQSeries datasets SCSQLOAD and SCSQAUTH, as well as one of the language-specific datasets such as SCSQANLE. See the MQSeries 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

C applications on Solaris

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 "Language compilers" on page 426 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/mgm/lib.
- Link with the AMI library.

To do this, use the -1 flag, more specifically -1amt.

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 Sun Solaris" on page 443).

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.

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

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

For the MQSeries 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 and Windows NT operating systems. See "Language compilers" on page 426 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 MQSeries. 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 MQSeries installed, and you have installed AMI under the MQSeries 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 MQSeries runtime environment. (This will be the case if you installed MQSeries using the documented method.)

C applications on Windows

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 38
- "Reference information for the C high-level interface" on page 39

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 45 amTerminate page 60

Sending messages

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

amSendMsgpage 56amSendRequestpage 57amSendResponsepage 58

Receiving messages

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

amReceiveMsgpage 49amReceiveRequestpage 53amBrowseMsgpage 42

File transfer

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

amSendFilepage 55amReceiveFilepage 47

Publish/subscribe

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

amPublishpage 46amSubscribepage 59amUnsubscribepage 61amReceivePublicationpage 51

Transaction support

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

amBeginpage 41amCommitpage 44amBackoutpage 40

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" on page 497).

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, policyName, AMSTR PAMLONG pCompCode, 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 MQSeries 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
            rcvMsqName,
  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. The locking

feature is not available on OS/390.

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 90). 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.

C high-level interface

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

1

ı

ı

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, 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 (MQSeries).

```
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 90).

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 90). The message object is reset implicitly before

the receive takes place.

pCompCode Completion code (output).

C high-level interface

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,
           rcvMsgName,
  AMSTR
  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 90). The message object is implicitly reset before

the receive takes place.

pCompCode Completion code (output).

pReason Reason code (output).

Usage notes

I

ı

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

C high-level interface

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 90).

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 90).

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 90). 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.

C high-level interface

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 47 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 90).

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 90).

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).

amSendRequest

Function to send a request message.

```
AMBOOL amSendRequest(
 AMHSES
           hSession,
 AMSTR
           senderName,
 AMSTR
           policyName,
 AMSTR
          responseName,
 AMLONG
           dataLen,
 PAMBYTE pData,
           sndMsgName,
 AMSTR
 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 90).

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).

amSendResponse

Function to send a response to a request message.

AMBOOL amSendResponse(AMHSES hSession, AMSTR senderName, AMSTR policyName, rcvMsgName, AMSTR AMLONG dataLen, PAMBYTE pData, sndMsgName, AMSTR 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 90).

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).

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 90).

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 MQSeries 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).

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, pReason);
```

Parameters

phSession A pointer to the session handle returned by amInitialize

(input/output).

The name of a policy (input). If specified as NULL, the system policyName

default policy (constant: AMSD_POL) is used.

pCompCode Completion 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 90).

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 MQSeries

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 64
Message	page 66
Sender	page 68
Receiver	page 69
Distribution list	page 70
Publisher	page 71
Subscriber	page 72
Policy	page 73

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 74.

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 79
amSesOpen	page 88
amSesClose	page 79
amSesDelete	page 83

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 80
amSesCreateSender	page 82
amSesCreateReceiver	page 81
am Ses Create Dist List	page 80
amSesCreatePublisher	page 81
amSesCreateSubscriber	page 82
amSesCreatePolicy	page 80

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).

am Ses Get Message Handle	page 86
am Ses Get Sender Handle	page 88
am Ses Get Receiver Handle	page 87
am Ses Get Dist List Handle	page 85
am Ses Get Publisher Handle	page 87
am Ses Get Subscriber Handle	page 88
am Ses Get Policy Handle	page 87

C object interface overview

Delete objects

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

amSesDeleteMessagepage 83amSesDeleteSenderpage 85amSesDeleteReceiverpage 84amSesDeleteDistListpage 83amSesDeletePublisherpage 84amSesDeleteSubscriberpage 85amSesDeletePolicypage 84

Transactional processing

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

amSesBeginpage 78amSesCommitpage 79amSesRollbackpage 89

Error handling

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

amSesClearErrorCodespage 78amSesGetLastErrorpage 86

Message interface functions

A message object encapsulates an MQSeries 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 94
amMsgGetCorrelId	page 94
amMsgGetElementCCSID	page 96
amMsgGetEncoding	page 96
amMsgGetFormat	page 98
am Msg Get Group Status	page 98
amMsgGetMsgId	page 99
amMsgGetName	page 100
am Msg Get Report Code	page 101
amMsgGetType	page 102

Set values

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

amMsgSetCCSID	page 104
am Msg Set Correl Id	page 104
amMsgSetElementCCSID	page 105
amMsgSetEncoding	page 105
amMsgSetFormat	page 106
amMsgSetGroupStatus	page 106

Reset values

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

amMsgResetpage 103

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.

am Msg Get Data Length	page 95
am Msg Get Data Off set	page 95
am Msg Set Data Off set	page 104
am Msg Read Bytes	page 103
amMsgWriteBytes	page 107

C object interface overview

Publish/subscribe topics

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

amMsgAddTopicpage 92amMsgDeleteTopicpage 94amMsgGetTopicpage 102amMsgGetTopicCountpage 103

Publish/subscribe filters

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

amMsgAddFilterpage 91amMsgDeleteFilterpage 93amMsgGetFilterpage 97amMsgGetFilterCountpage 97

Publish/subscribe name/value elements

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

amMsgAddElement page 91
amMsgDeleteElement page 92
amMsgGetElement page 95
amMsgGetElementCount page 96
amMsgDeleteNamedElement page 93
amMsgGetNamedElement page 100
amMsgGetNamedElementCount page 101

Error handling

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

amMsgClearErrorCodespage 92amMsgGetLastErrorpage 99

Publish/subscribe helper macros

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

AmMsgAddStreamNamepage 108AmMsgGetPubTimestamppage 108AmMsgGetStreamNamepage 109

Sender interface functions

A sender object encapsulates an MQSeries object descriptor (MQOD) structure for sending a message.

Open and close

Functions to open and close the sender service.

amSndOpenpage 112amSndClosepage 110

Send

Function to send a message.

amSndSendpage 113amSndSendFilepage 114

Get values

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

amSndGetCCSIDpage 111amSndGetEncodingpage 111amSndGetNamepage 112

Error handling

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

amSndClearErrorCodesamSndGetLastErrorpage 110

Receiver interface functions

A receiver object encapsulates an MQSeries object descriptor (MQOD) structure for receiving a message.

Open and close

Functions to open and close the receiver service.

amRcvOpen page 121 amRcvClose page 119

Receive and browse

Functions to receive or browse a message.

amRcvReceivepage 122amRcvReceiveFilepage 124amRcvBrowsepage 115amRcvBrowseSelectpage 117

Get values

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

amRcvGetDefnTypepage 119amRcvGetNamepage 120amRcvGetQueueNamepage 121

Set values

Function to set the queue name of the receiver service.

amRcvSetQueueName page 125

Error handling

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

amRcvClearErrorCodespage 118amRcvGetLastErrorpage 120

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 128 amDstClose page 126

Send

Function to send a message to the distribution list.

amDstSend page 129 amDstSendFile page 130

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 127 amDstGetSenderCountpage 128 amDstGetSenderHandle page 128

Error handling

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

amDstClearErrorCodespage 126 amDstGetLastError page 127

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.

amPubOpenpage 133amPubClosepage 131

Publish

Function to publish a message.

amPubPublish page 134

Get values

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

amPubGetCCSID page 131amPubGetEncoding page 132amPubGetName page 133

Error handling

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

amPubClearErrorCodespage 131amPubGetLastErrorpage 132

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 138 amSubClose page 135

Broker messages

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

amSubSubscribe page 140 amSubUnsubscribe page 141 amSubReceive page 139

Get values

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

amSubGetCCSID page 135 amSubGetDefnType page 136 amSubGetEncoding page 136 amSubGetName page 137 amSubGetQueueName page 138

Set value

Function to set the queue name of the subscriber service.

amSubSetQueueName page 139

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 135 amSubGetLastError page 137

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 143amPolGetWaitTime page 143

Set value

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

amPolSetWaitTime page 143

Error handling

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

amPolClearErrorCodespage 142amPolGetLastErrorpage 142

High-level functions

Each high-level function described in "Chapter 3. The C high-level interface" on page 37 calls a number of the object interface functions, as shown in the following

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

High-level function	Equivalent object interface calls 1
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

C object interface overview

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

High-level function	Equivalent object interface calls 1
Note:	
1. 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.	

C object interface overview

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 78
Message	page 90
Sender	page 110
Receiver	page 115
Distribution list	page 126
Publisher	page 131
Subscriber	page 135
Policy	page 142

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" on page 497).

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 (MQSeries) connection object encapsulates a single MQSeries 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 MQSeries 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 MQSeries. The unit of work can subsequently be committed by **amSesCommit**, or backed out by **amSesRollback**. It should be used only when MQSeries 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 (MQSeries).

```
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).

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).

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).

C session interface

amSesCreateSender

PAMLONG

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

AMHSND amSesCreateSender(
AMHSES hSess,
AMSTR name,
PAMLONG pCompCode,

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).

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).

C session interface

amSesDeletePolicy

```
Deletes a policy object.

AMBOOL amSesDeletePolicy(
AMHSES hSess,
PAMHPOL phPolicy,
PAMLONG pCompCode,
```

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).

pReason);

amSesDeletePublisher

PAMLONG

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).

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).

C session interface

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 an MQSeries 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).

Page 200 and a (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).

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 MQSeries local queue manager library cannot be loaded, the MQSeries client library is loaded. (On OS/390, client connections are not supported so applications must use a local queue manager.) The connection to the underlying message transport (MQSeries) 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).

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).

Message interface functions

A message object encapsulates an MQSeries 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:

default queue manager CCSID

correlationId all zeros dataLength zero dataOffset zero elementCount zero

encoding AMENC_NATIVE format AMFMT_STRING

AMGRP_MSG_NOT_IN_GROUP groupStatus

topicCount

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 103) 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 91
am Msg Delete Element	page 92
amMsgGetElement	page 95
amMsgGetElementCount	page 96
am Msg Delete Name d Element	page 93
amMsgGetNamedElement	page 100
amMsgGetNamedElementCount	
	page 101

amMsgAddTopic page 92 amMsgDeleteTopic page 94 amMsgGetTopic page 102 amMsgGetTopicCount page 103

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 MQSeries Integrator Version 2.0 Programming Guide.

pCompCode Completion 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).

Reason code (output).

pReason

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).

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).

amMsgGetCorrelld

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).

p0ffset 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

pCompCode

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

Completion code (output).

pReason Reason code (output).

96

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).

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).

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 an MQSeries 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.

amMsgGetMsgld

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).

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).

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). The following values can be returned:

AMFB_EXPIRATION AMFB_COA AMFB_COD AMFB_ERROR

pCompCode Completion 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,
          pCompCode,
  PAMLONG
  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).

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 90).

```
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).

amMsgSetCorrelld

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).

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 103 and "amMsgWriteBytes" on page 107.

```
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).

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).

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).

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).

amMsgWriteBytes

Writes the specified number of data bytes into the message object, starting at the current data offset. See "amMsgSetDataOffset" on page 104.

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).

pCompCode The stream name (input).

Completion code (output).

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).

Sender interface functions

A sender object encapsulates an MQSeries object descriptor (MQOD) structure. This represents an MQSeries 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 and policies" on page 469). 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);
```

The sender handle returned by amSesCreateSender (input). hSender

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 an MQSeries reason

code.

C sender interface

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, buffLen, AMLONG 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).

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(
           hSender,
  AMHSND
  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).

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);
```

Parameters

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 124 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 an MQSeries object descriptor (MQOD) structure. This represents a local MQSeries 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 MQSeries Application Programming Guide for a full description of the browse options.

```
AMBOOL amRcvBrowse(
  AMHRCV
            hReceiver,
  AMHP0L
            hPolicy,
  AMLONG
            options,
  AMLONG
            buffLen,
  PAMI ONG
            pDataLen,
  PAMBYTE
            pData,
  AMHMSG
            hRcvMsg,
  AMHSND
            hSender,
  PAMLONG
            pCompCode,
  PAMLONG
            pReason);
hReceiver
               The receiver handle returned by amSesCreateReceiver (input).
               The handle of a policy (input). If specified as
hPolicy
               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 + AMBRW_NEXT)
               AMBRW_LOCK_NEXT
                                      (AMBRW_LOCK + AMBRW_FIRST)
               AMBRW LOCK FIRST
               AMBRW LOCK CURRENT
                                      (AMBRW LOCK + AMBRW CURRENT)
               AMBRW UNLOCK
```

C receiver interface

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. The locking feature is not available on OS/390.

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 MQSeries 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,
            pCompCode,
  PAMLONG
  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

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. The locking feature is not available on OS/390.

hSe1Msg

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 104)

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).

C receiver interface

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).

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).

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 an MQSeries 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).

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).

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 78) 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)

C receiver interface

 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 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(hReceiver, AMHRCV **AMHPOL** hPolicy, AMHLONG options, hSelMsg, AMHMSG 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).

hSe1Msg 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 104) 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).

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,
 AMHP0L
           hPolicy,
 PAMLONG
           pCompCode,
 PAMLONG
           pReason);
```

hDistList The distribution list handle returned by amSesCreateDistList

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).

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 an MQSeries 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).

C distribution list interface

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).

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).

C distribution list interface

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);
```

Parameters

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 124 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).

C publisher interface

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).
```

amPubGetLastError

pReason

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).

Reason code (output).

pReason2 A secondary reason code (output). Not returned if specified as

> NULL. If pReason indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, pReason2 gives an MQSeries 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).

C publisher interface

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,
  AMHP0L
           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).

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).

C subscriber interface

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).

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 an MQSeries 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).

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.

pCompCode The queue name (output).

Completion code (output).

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 90).

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).

C subscriber interface

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 90) 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).

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 90) 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).

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(
 AMHP0L
           hPolicy,
 PAMLONG
           pCompCode,
 PAMLONG
           pReason);
```

The policy handle returned by amSesCreatePolicy (input). hPolicy

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 an MQSeries 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.

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).

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).

C policy interface

Part 3. The C++ interface

Chapter 6. Using the Application Messaging Interface in C++	Create objects
Structure of the AMI	Transactional processing 169
Base classes	Error handling
Interface and helper classes	AmMessage
Exception classes	Get values
Using the repository	Set values
System default objects	Reset values
Writing applications in C++	Read and write data
Creating and opening objects	Publish/subscribe topics
Deleting objects	Publish/subscribe filters
Sending messages	Publish/subscribe name/value elements 172
Sample program	Error handling
Receiving messages	AmSender
Sample program	Open and close
Request/response messaging	Send
Sample programs	Send file
File transfer	Get values
Publish/subscribe messaging 156	Error handling
Sample programs	AmReceiver
Using AmElement objects	Open and close
Error handling	Receive and browse
Transaction support	Receive file
Sending group messages	Get values
Other considerations	Set value
Multithreading	Error handling
Using MQSeries with the AMI 159	AmDistributionList
Field limits	Open and close
Building C++ applications	Send
AMI include files	Send file
C++ applications on AIX	Get values
Preparing C++ programs on AIX	Error handling
Running C++ programs on AIX 162	AmPublisher
C++ applications on AS/400	Open and close
Preparing C++ programs on AS/400 162	Publish
Running C++ programs on AS/400 163	Get values
C++ applications on HP-UX	
	Error handling
Preparing C++ programs on HP-UX 163	AmSubscriber
Running C++ programs on HP-UX 163	Open and close
C++ applications on Solaris	Broker messages
Preparing C++ programs on Solaris 164	Get values
Running C++ programs on Solaris 164	Set value
C++ applications on Windows 165	Error handling
Preparing C++ programs on Windows 165	AmPolicy
Running C++ programs on Windows 165	Policy management
	Error handling
Chapter 7. C++ interface overview	Helper classes
Base classes	AmBytes
Helper classes	AmElement
Exception classes	AmObject
AmSessionFactory	AmStatus
Constructor	AmString
Session factory management	Exception classes
Create and delete session	AmException
AmSession	AmErrorException
Session management	AmWarningException
	0 1 101

Chapter 8. C++ interface reference			18	83	getFilter
Base classes					getFilterCount
Helper classes					getFormat
Exception classes					getGroupStatus
AmSessionFactory					getLastErrorStatus
AmSessionFactory					getMessageId
createSession					getName
deleteSession					getNamedElement
getFactoryName					getNamedElementCount
getLocalHost					getReportCode
getRepository					getTopic
getTraceLevel					getTopicCount
getTraceLocation					9 .
setLocalHost					getType
setRepository					readBytes
setTraceLevel					reset
					setCCSID
setTraceLocation					setDataOffset
				-	setElementCCSID
begin					
clearErrorCodes					setEncoding
close			. 18	-	setFormat
commit					setGroupStatus
createDistributionList					writeBytes
createMessage					AmSender
createPolicy					clearErrorCodes
createPublisher					close
createReceiver					enableWarnings
createSender					getCCSID
createSubscriber					getEncoding
deleteDistributionList					getLastErrorStatus
deleteMessage					getName
deletePolicy					open
deletePublisher					send
deleteReceiver					sendFile
deleteSender					AmReceiver
deleteSubscriber					browse
enableWarnings					clearErrorCodes
getLastErrorStatus					close
getName					enableWarnings
getTraceLevel					getDefinitionType
getTraceLocation					getLastErrorStatus
open					getName
rollback					getQueueName
AmMessage					open
addElement					receive
addFilter					receiveFile
addTopic					setQueueName
clearErrorCodes					AmDistributionList
deleteElement					clearErrorCodes 205
deleteFilter					close
deleteNamedElement					enableWarnings
deleteTopic					getLastErrorStatus
enableWarnings					getName
getCCSID					getSender
getCorrelationId					getSenderCount 205
getDataLength					open
getDataOffset					send
getElement					sendFile
getElementCCSID					AmPublisher
getElementCount					clearErrorCodes 207
getEncoding			. 19	93	close

2227	
getCCSID	destructor
	operators
getEncoding	oad
getLastErrorStatus	split
getName	strip
open	ength
figure 4	ext
	runcate
	Exception
	getClassName
· ·	getCompletionCode
	getMethodName
	getReasonCode
0 11	getSource
	coString
	ErrorException
· ·	getClassName
	getCompletionCode
	getMethodName
· ·	getReasonCode
	getSource
	coString
	WarningException
	getClassName
enableWarnings 212	getCompletionCode
getLastErrorStatus	getMethodName
getName	getReasonCode
	getSource
= -	oString
AmBytes	
cmp	
constructors	
сру	
dataPtr	
destructor	
length	
operators	
pad	
AmElement	
AmElement	
getName	
gctivatic	
getValue	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217 AmStatus 217	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217 AmStatus 217 getCompletionCode 217	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217 AmStatus 217 getCompletionCode 217 getReasonCode 217	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217 AmStatus 217 getCompletionCode 217 getReasonCode 217 getReasonCode2 217	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217 AmStatus 217 getCompletionCode 217 getReasonCode 217	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217 AmStatus 217 getCompletionCode 217 getReasonCode 217 getReasonCode2 217 toString 217 AmString 218	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217 AmStatus 217 getCompletionCode 217 getReasonCode 217 getReasonCode2 217 toString 217	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217 AmStatus 217 getCompletionCode 217 getReasonCode 217 getReasonCode2 217 toString 217 AmString 218	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217 AmStatus 217 getCompletionCode 217 getReasonCode 217 getReasonCode2 217 toString 217 AmString 218 cat 218	
getValue 215 getVersion 215 setVersion 215 toString 215 AmObject 216 clearErrorCodes 216 getLastErrorStatus 216 getName 216 AmStatus 217 getCompletionCode 217 getReasonCode 217 getReasonCode2 217 toString 217 AmString 218 cat 218 cmp 218	

. . . 218 . . . 219 . . . 219 . . . 219 . . . 219 . . . 219 . . . 219 . . . 219 . . . 220 . . . 220 . . 220 . . . 220 . . . 220 . . . 220 . . . 220 . . . 221 . . . 221 . . . 221 . . . 221 . . . 221 . . . 221 . . . 221 . . . 222 . . . 222 . . . 222 . . . 222 . . . 222 . . . 222 . . . 222

Chapter 6. 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 151
- "Building C++ applications" on page 161

Note that the term *object* is used in this book in the object-oriented programming sense, not in the sense of MQSeries '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 an MQSeries queue) to which messages are sent.

AmReceiver This is a service that represents a source (such as

an MQSeries 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.

Structure of the AMI

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 470).

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 470).

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 Message ID. Binary data, such as Message Id 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 171.

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 195) and rebuild it each time.

Sample program

For more details, refer to the SendAndForget.cpp sample program (see "Sample programs for AS/400, UNIX, and Windows" on page 461).

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 AS/400, UNIX, and Windows" on page 461).

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 152, 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 153, 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 153).

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 AS/400, UNIX, and Windows" on page 461).

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:

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 MQSeries Integrator Version 2" on page 458).

Sample programs

For more details, refer to the Publisher.cpp and Subscriber.cpp sample programs (see "Sample programs for AS/400, UNIX, and Windows" on page 461).

Using AmElement objects

Publish/subscribe brokers (such as MQSeries 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", "RegUpdate"); 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 MQSeries 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.

Writing applications in C++

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

- · MQSeries 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 MQSeries 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 26

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 (MQSeries) 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 MQSeries and a transport error (AMRC_TRANSPORT_ERR) will be reported to the application.

Using MQSeries with the AMI

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

Writing applications in C++

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 MQSeries, the underlying message transport. See the rules for naming MQSeries objects in the MQSeries Application Programming Guide.

Building C++ applications

I

I

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:

OMOMAMI/H (AS/400)/amt/inc (UNIX) \amt\include (Windows)

See "Directory structure" on page 430 (AIX), page 433 (AS/400), page 438 (HP-UX), page 445 (Solaris), or page 448 (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 AS/400" on page 162
- "C++ applications on HP-UX" on page 163
- "C++ applications on Solaris" on page 164
- "C++ applications on Windows" on page 165

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 "Language compilers" on page 426 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 xIC 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.

C++ applications on AIX

- 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 -1 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 428).

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 MQSeries in your runtime environment. To do this, make the AMI MQSeries runtime binding stubs available, to allow AMI to load MQSeries libraries dynamically. For the non-threaded MQSeries Server library, perform:

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

For the non-threaded MQSeries Client library, perform:

ln -s /usr/mgm/lib/amtcmgic /usr/lib/amtcmgic

For the threaded MQSeries Server library, perform:

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

For the threaded MQSeries Client library, perform:

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

C++ applications on AS/400

This section explains what you have to do to prepare and run your C++ programs on the AS/400 system. See "Language compilers" on page 426 for the compilers supported by the AMI.

Preparing C++ programs on AS/400

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.

C++ applications on AS/400

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 AS/400

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 HP-UX

1

1

I

1

ı I

ı

This section explains what you have to do to prepare and run your C++ programs on the HP-UX operating system. See "Language compilers" on page 426 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 -1 flag, more specifically -lamtCpp.

For example, to compile the C++ program mine.cpp into an executable called mine:

```
aCC +DAportable -W1,+b,:,-L/opt/mgm/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 436).

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

C++ applications on HP-UX

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 MQSeries in your runtime environment. To do this, make the AMI MQSeries runtime binding stubs available, to allow AMI to load MQSeries libraries dynamically. For the non-threaded MQSeries Server library, perform:

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

For the non-threaded MQSeries Client library, perform:

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

For the threaded MQSeries Server library, perform:

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

For the threaded MQSeries Client library, perform:

```
ln -s /opt/mgm/lib/amtcmgic r /usr/lib/amtcmgic r
```

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

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 "Language compilers" on page 426 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/mgm/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 -1 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 Sun Solaris" on page 443).

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 MQSeries in your runtime environment. To do this, make the AMI MQSeries runtime binding stubs available, to allow AMI to load MQSeries libraries dynamically. For the MQSeries Server library, perform:

ln -s /opt/mgm/lib/amtcmgm /usr/lib/amtcmgm

For the MQSeries 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 and Windows NT operating systems. See "Language compilers" on page 426 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 MQSeries. 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 MQSeries installed, and you have installed AMI under the MQSeries directory structure, it is likely that the PATH has already been set up for you.

You also need access to the C libraries and MQSeries in your runtime environment. (This will be the case if you installed MQSeries using the documented method.)

C++ applications on Windows

Chapter 7. 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

AmSessionFactory	page 168
AmSession	page 169
AmMessage	page 171
AmSender	page 173
AmReceiver	page 174
AmDistributionList	page 175
AmPublisher	page 176
AmSubscriber	page 177
AmPolicy	page 178

Helper classes

AmBytes	page 179
AmElement	page 179
AmObject	page 179
AmStatus	page 179
AmString	page 180

Exception classes

AmException	page 181
AmErrorException	page 181
AmWarningExcpetion	page 181

AmSessionFactory

The AmSessionFactory class is used to create AmSession objects.

Constructor

Constructor for AmSessionFactory.

AmSessionFactory page 184

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	184
getLocalHost	page	184
getRepository	page	184
getTraceLevel	page	184
getTraceLocation	page	184
setLocalHost	page	185
setRepository	page	185
setTraceLevel	page	185
setTraceLocation	page	185

Create and delete session

Methods to create and delete an AmSession object.

create Sessionpage 184 deleteSession page 184

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

open	page 189
close	page 186
getName	page 189
getTraceLevel	page 189
getTraceLocation	page 189

Create objects

Methods to create AmMessage, AmSender, AmReceiver, AmDistributionList, AmPublisher, AmSubscriber, and AmPolicy objects.

createMessage	page 187
createSender	page 187
createReceiver	page 187
create Distribution List	page 186
createPublisher	page 187
createSubscriber	page 187
createPolicy	page 187

Delete objects

Methods to delete AmMessage, AmSender, AmReceiver, AmDistributionList, AmPublisher, AmSubscriber, and AmPolicy objects.

deleteMessage	page 188
deleteSender	page 188
deleteReceiver	page 188
deleteDistributionList	page 188
deletePublisher	page 188
deleteSubscriber	page 188
deletePolicy	page 188

Transactional processing

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

begin	page 186
commit	page 186
rollback	page 189

C++ interface overview

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clear Error Codespage 186 enableWarnings page 188 getLastErrorStatuspage 189

AmMessage

An AmMessage object encapsulates an MQSeries 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.

page 192
page 192
page 193
page 193
page 193
page 194
page 194
page 194
page 195
page 195

Set values

Methods to set the coded character set ID, correlation ID, format and group status of the message object.

setCCSID	page 196
setCorrelationId	page 196
setElementCCSID	page 196
setEncoding	page 196
setFormat	page 197
setGroupStatus	page 197

Reset values

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

reset page 195

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 192
getDataOffset	page 192
set Data Off set	page 196
readBytes	page 195
writeBytes	page 197

C++ interface overview

Publish/subscribe topics

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

addTopic page 191 deleteTopic page 192 getTopic page 195 getTopicCount page 195

Publish/subscribe filters

Methods to manipulate filters for content-based publish/subscribe.

addFilter page 191 deleteFilter page 191 getFilter page 193 getFilterCount page 193

Publish/subscribe name/value elements

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

addElement page 191 deleteElement page 191 getElement page 193 getElementCount page 193 deleteNamedElement page 192 getNamedElement page 194 getNamedElementCount page 194

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 191 enableWarnings page 192 getLastErrorStatus page 194

AmSender

An **AmSender** object encapsulates an MQSeries object descriptor (MQOD) structure.

Open and close

Methods to open and close the sender service.

page 199 open close page 198

Send

Method to send a message.

send page 199

Send file

Method to send data from a file

sendFile page 200

Get values

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

getCCSID page 198 getEncoding page 198 getName page 199

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 198 enableWarnings page 198 getLastErrorStatuspage 199

AmReceiver

An **AmReceiver** object encapsulates an MQSeries object descriptor (MQOD) structure.

Open and close

Methods to open and close the receiver service.

page 203 open close page 202

Receive and browse

Methods to receive or browse a message. page 203 receive

browse page 201

Receive file

Method to receive file message data into a file.

receiveFile page 204

Get values

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

getDefinitionType page 202 getName page 203 getQueueName page 203

Set value

Method to set the queue name of the receiver service.

setQueueName page 204

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 202 enableWarnings page 202 getLastErrorStatus page 203

AmDistributionList

An AmDistributionList object encapsulates a list of AmSender objects.

Open and close

Methods to open and close the distribution list service.

open page 205 close page 205

Send

Method to send a message to the distribution list.

page 206 send

Send file

Method to send date from a file to the each sender defined in the distribution list.

sendFile page 206

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 205 getSenderCount page 205 getSender page 205

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 205 enableWarnings page 205 getLastErrorStatus page 205

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.

page 208 open close page 207

Publish

Method to publish a message.

publish page 208

Get values

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

getCCSID page 207 getEncoding page 207 getName page 207

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 207 enableWarnings page 207 getLastErrorStatuspage 207

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.

page 210 open close page 209

Broker messages

Methods to subscribe to a broker, remove a subscription, and receive a publication from the broker.

subscribe page 211 unsubscribe page 211 receive page 210

Get values

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

getCCSID page 209 getDefinitionType page 209 getEncoding page 209 getName page 210 getQueueName page 210

Set value

Method to set the queue name of the subscriber service.

setQueueName page 210

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 209 enableWarnings page 209 getLastErrorStatus page 209

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 212 getWaitTime page 212 setWaitTime page 212

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 212 enableWarnings page 212 page 212 getLastErrorStatus

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 213
destructor	page 214
operators	page 214
cmp	page 213
сру	page 214
dataPtr	page 214
length	page 214
pad	page 214

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 215
getName	page 215
getValue	page 215
getVersion	page 215
setVersion	page 215
toString	page 215

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 216
getLastErrorStatus	page 216
getName	page 216

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 217
getCompletionCode	page 217

C++ interface overview

getReasonCode	page 217
getReasonCode2	page 217
toString	page 217

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 218
destructor	page 218
operators	page 219
cat	page 218
cmp	page 218
contains	page 218
сру	page 218
length	page 219
pad	page 219
split	page 219
strip	page 219
text	page 219
truncate	page 219

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 220
get Completion Code	page 220
getMethodName	page 220
getReasonCode	page 220
getSource	page 220
toString	page 220

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 221
get Completion Code	page 221
getMethodName	page 221
getReasonCode	page 221
getSource	page 221
toString	page 221

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 222
get Completion Code	page 222
getMethodName	page 222
getReasonCode	page 222
getSource	page 222
toString	page 222

C++ interface overview

Chapter 8. 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.

AmSessionFactory	page 184
AmSession	page 186
AmMessage	page 190
AmSender	page 198
AmReceiver	page 201
AmDistributionList	page 205
AmPublisher	page 207
AmSubscriber	page 209
AmPolicy	page 212

Helper classes

AmBytes	page 213
AmElement	page 215
AmObject	page 216
AmStatus	page 217
AmString	page 218

Exception classes

AmException	page 220
AmErrorException	page 221
AmWarningException	page 222

AmSessionFactory

The AmSessionFactory class is used to create AmSession objects.

AmSessionFactory

Constructors for an AmSessionFactory.

```
AmSessionFactorv():
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 (AS/400, UNIX, and Windows)" on page 450 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

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.

setTraceLevel

Sets the trace level for the AmSessionFactory.

void setTraceLevel(int level);

leve1

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 (MQSeries) connection object encapsulates a single MQSeries 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 MQSeries MQCONN or MQDISC requests (or their equivalent in the MQSeries 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 MQSeries. 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);

warnings0n 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 MQSeries and the MQSeries local queue manager library cannot be loaded, the MQSeries 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 MQSeries 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 AMFMT_STRING format

AMGRP_MSG_NOT_IN_GROUP groupStatus

reportCode AMFBP_NONE

topicCount zero

AMMT_DATAGRAM type

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 195) 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:

page 191
page 191
page 193
page 193
page 192
page 194
page 194
page 191
page 192
page 195
page 195

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);
```

The name of the element to be deleted. name

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(AMB00L warnings0n);
```

warnings0n

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
```

C++ AmMessage

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 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,
          index):
  int
```

The name of the element to be returned. name

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();
```

The following values can be returned:

```
AMFB NONE
AMFB EXPIRATION
AMFB COA
AMFB_COD
AMFB_ERROR
```

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 190).

```
reset is overloaded: the options parameter is optional.
      reset(int options);
```

options A reserved field that must be set to zero.

C++ AmMessage

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 a null string; for example: myMessage.setCorrelationId(AmBytes(""));

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);

element CCSID 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).

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

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 an MQSeries object descriptor (MQOD) structure. This represents an MQSeries 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);

warnings0n 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.

C++ AmSender

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 204 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 an MQSeries object descriptor (MQOD) structure. This represents an MQSeries 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_LOCK_NEXT

AMBRW_LOCK_FIRST

AMBRW_LOCK_CURRENT

AMBRW_LOCK_CURRENT

AMBRW_LOCK_CURRENT

AMBRW_UNLOCK

AMBRW_UNLOCK

AMBRW_LOCK + AMBRW_CURRENT)
```

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 196) 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);

warnings0n

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 196) 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 196) 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);
```

warnings0n

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.

C++ AmDistributionList

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 204 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);

warnings0n

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();
```

C++ AmPublisher

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, &policy);

AmPolicy

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);
```

warnings0n

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();
```

C++ AmSubscriber

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 190).

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 196) 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 190) 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 190) 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(AMB00L warnings0n);

warnings0n

If set to AMB_TRUE, AmWarningExceptions will be raised for this

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.

A single byte used to create the new AmBytes object.

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.

C++ AmBytes

сру

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.
               The value of the element.
value
```

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);
              The version of the AmElement that is set. It can take the value
version
              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 MQSeries, the secondary reason code is an MQSeries 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;
```

A reference to the AmString object being compared. amString

stringData The NULL terminated string being compared.

constructors

Constructors for an AmString object.

```
AmString();
AmString(const AmString &amString);
AmString(const AMSTR stringData);
```

A reference to an AmString object used to create the new amString

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.

сру

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);
```

A reference to an AmString object being copied. amString

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 9. Using the Application Messaging	AMHRCRQ (receive request) 260
Interface in COBOL	AMHSNFL (send file)
Structure of the AMI	AMHSNMS (send message) 263
Using the repository	AMHSNRQ (send request) 264
System default objects	AMHSNRS (send response) 265
Writing applications in COBOL	AMHSB (subscribe)
Opening and closing a session	AMHTERM (terminate)
Sending messages	AMHUN (unsubscribe) 268
Using the message object	,
Sample programs	Chapter 11. COBOL object interface overview 269
Receiving messages	Session interface functions
Using the message object	Session management
Sample programs	Create objects
Request/response messaging	Get object handles
Request	Delete objects
Response	Transactional processing
Sample programs	Error handling
File transfer	Message interface functions
Publish/subscribe messaging	Get values
Publish	
Subscribe	Set values
	Reset values
Sample programs	Read and write data
Using name/value elements	Publish/subscribe topics
Example	Publish/subscribe filters
Error handling	Publish/subscribe name/value elements 273
Transaction support	Error handling
Sending group messages	Sender interface functions
	Open and close.
Multithreading	
Using MQSeries with the AMI 240	Get values
Field limits	Error handling
Building COBOL applications	Receiver interface functions
COBOL applications on OS/390 241	Open and close
AMI Copybooks 241	Receive and browse
Preparing COBOL programs on OS/390 241	Get values
Running COBOL programs on OS/390 242	Set values
	Error handling
Chapter 10. The COBOL high-level interface 243	Distribution list interface functions
Overview of the COBOL high-level interface 244	Open and close
Initialize and terminate	Send
Sending messages 244	Get values
Receiving messages 244	Error handling
File transfer	Publisher interface functions
Publish/subscribe 244	Open and close
Transaction support 244	Publish
Reference information for the COBOL high-level	Get values
interface	Error handling
AMHBACK (backout) 247	Subscriber interface functions
AMHBEGIN (begin) 248	Open and close
AMHBRMS (browse message) 249	Broker messages
AMHCMIT (commit)	Get values
AMHINIT (initialize)	Set value
AMHPB (publish)	Error handling
AMHRCFL (receive file)	Policy interface functions
AMHRCMS (receive message) 256	Get values
AMHRCPB (receive publication)	Set value

Error handling	. 279	AMMSGTRC (get report code)	 . 309
High-level functions	. 280	AMMSGTTO (get topic)	 . 309
		AMMSGTTC (get topic count)	 . 310
Chapter 12. COBOL object interface reference	283	AMMSGTTY (get type)	 . 310
Session interface functions	. 284	AMMSREBY (read bytes)	
AMSEBG (begin)		AMMSRS (reset)	 . 311
AMSECLEC (clear error codes)		AMMSSTCC (set CCSID)	 . 312
AMSECL (close)		AMMSSTCI (set correl ID)	 . 312
AMSECM (commit)	. 285	AMMSSTDO (set data offset)	 . 312
AMSECR (create)		AMMSSELC (set element ccsid)	
AMSECRDL (create distribution list)		AMMSSTEN (set encoding)	 . 313
AMSECRMS (create message)		AMMSSTFO (set format)	
AMSECRPO (create policy)		AMMSSTGS (set group status)	
AMSECRPB (create publisher)		AMMSWRBY (write bytes)	
AMSECRRC (create receiver)		Sender interface functions	
AMSECRSN (create sender)		AMSNCLEC (clear error codes)	
AMSECRSB (create subscriber)		AMSNCL (close)	
AMSEDL (delete)		AMSNGTCC (get CCSID)	
AMSEDLDL (delete distribution list)		AMSNGTEN (get encoding)	
AMSEDLMS (delete message)		AMSNGTLE (get last error)	
AMSEDLPO (delete policy)		AMSNGTNA (get name)	 318
AMSEDLPB (delete publisher)		AMSNOP (open)	
AMSEDLR (delete receiver)		AMSNSN (send)	
		AMSNSNFL (send file)	
AMSEDLSN (delete sender)		Usage notes	 320
AMSECLIDI (cet distribution list handle)		Receiver interface functions	 320
AMSEGHDL (get distribution list handle)		AMRCBR (browse)	
AMSECTIMS (get last error codes)		Usage notes	
AMSEGHMS (get message handle)			
AMSEGHPO (get policy handle)		AMRCBRSE (browse selection message)	
AMSEGHPB (get publisher handle)		Usage notes	 . 323
AMSEGHRC (get receiver handle)		AMRCCLEC (clear error codes)	
AMSEGHSN (get sender handle)		AMRCCI (close)	
AMSEGHSB (get subscriber handle)		AMRCGTDT (get definition type)	
AMSEOP (open)		AMRCGTLE (get last error)	
AMSERB (rollback)		AMRCGTNA (get name)	
Message interface functions		AMRCGTQN (get queue name)	
AMMSADEL (add element)		AMRCOP (open)	 . 328
AMMSADFI (add filter)		AMRCRC (receive)	 . 329
AMMSADTO (add topic)	. 299	Usage notes	
AMMSCLEC (clear error codes)		AMRCRCFL (receive file)	
AMMSDEEL (delete element)		AMRCSTQN (set queue name)	
AMMSDEFI (delete filter)		Distribution list interface functions	
AMMSDENE (delete named element)		AMDLCLEC (clear error codes)	
AMMSDETO (delete topic)		AMDLCL (close)	
AMMSGELC (get element CCSID)		AMDLGTLE (get last error)	
AMMSGTCC (get CCSID)	. 301	AMDLGTNA (get name)	
AMMSGTCI (get correl ID)		AMDLGTSC (get sender count)	
AMMSGTDL (get data length)	. 302	AMDLGTSH (get sender handle)	 . 334
AMMSGTDO (get data offset)		AMDLOP (open)	 . 334
AMMSGTEL (get element)		AMDLSN (send)	 . 335
AMMSGTEC (get element count)		AMDLSNFL (send file)	 . 335
AMMSGTEN (get encoding)		Usage notes	 . 336
AMMSGTFC (get filter count)		Publisher interface functions	
AMMSGTFI (get filter)		AMPBCLEC (clear error codes)	
AMMSGTFO (get format)		AMPBCL (close)	
AMMSGTGS (get group status)		AMPBGTCC (get CCSID)	
AMMSGTLE (get last error)		AMPBGTEN (get encoding)	
AMMSGTMI (get message ID)		AMPBGTLE (get last error)	
AMMSGTNA (get name)		AMPBGTNA (get name)	
AMMSGTNE (get named element)		AMPBOP (open)	
AMMSGTNC (get named element count)		AMPBPB (publish)	

Subscriber interface functions				341
AMSBCLEC (clear error codes)				341
AMSBCL (close)				341
AMSBGTCC (get CCSID)				342
AMSBGTDT (get definition type)				342
AMSBGTEN (get encoding) .				343
AMSBGTLE (get last error)				343
AMSBGTNA (get name)				344
AMSBGTQN (get queue name)				344
AMSBOP (open)				345
AMSBRC (receive)				345
AMSBSTQN (set queue name).				346
AMSBSB (subscribe)				346
AMSBUN (unsubscribe)				347
Policy interface functions				348
AMPOCLEC (clear error codes)				348
AMPOGTLE (get last error) .				348
AMPOGTNA (get name)				349
AMPOGTWT (get wait time) .				349
AMPOSTWT (set wait time) .				350

Chapter 9. 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 MQSeries users.

This chapter describes the following:

- "Structure of the AMI"
- "Writing applications in COBOL" on page 230
- "Building COBOL applications" on page 241

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 MQSeries '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 an MQSeries

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 <code>MessageID</code>), or to change the value of one or more properties before sending the message (for instance, the <code>Format</code>). 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,

Structure of the AMI

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 10. The COBOL high-level interface" on page 243. An overview of the object interface is given in "Chapter 11. COBOL object interface overview" on page 269, with reference information in "Chapter 12. COBOL object interface reference" on page 283.

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 470).

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 470).

Structure of the AMI

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" on page 509). 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 280).

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 252):

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 267):

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 263). 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 <code>MessageID</code>, 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 272 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 311) 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 OS/390" on page 464).

Receiving messages

Use the AMHRCMS high-level function (page 256) 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 OS/390" on page 464).

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 264) 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 260) 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 232.

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 265):

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 232. 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 OS/390" on page 464).

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 MQSeries Integrator Version 2" on page 458).

For more information, refer to the MQSeries Publish/Subscribe User's Guide.

Use the AMHPB high-level function (page 253) 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 266) 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 268). 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 258). 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 232.

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 OS/390" on page 464).

Using name/value elements

Publish/subscribe brokers (such as MQSeries 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
                                  PIC X(8) VALUE 'COEL
15 AMELEM-STRUCID
   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
                                  PIC S9(9) BINARY VALUE 0.
 15 AMELEM-GROUP-OFFSET
   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 MQSeries 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.

MQSeries messages are the only resource

This is supported under OS/390 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
{\tt 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 26.

Other considerations

You should consider the following when writing your applications:

- Multithreading
- Using MQSeries with the AMI
- Field limits

Multithreading

Multithreading is not supported for COBOL applications running on OS/390.

Using MQSeries with the AMI

You must not mix MQSeries 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 MQSeries, the underlying message transport. See the rules for naming MQSeries objects in the MQSeries Application Programming Guide.

Building COBOL applications

The Application Messaging Interface for COBOL is available only on the OS/390 operating system.

COBOL applications on OS/390

This section explains what you have to do to prepare and run your COBOL programs on the OS/390 operating system. See "Language compilers" on page 426 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 MQSeries for OS/390 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 OS/390

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: If you are using COBOL, you should 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 MQSeries for OS/390 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)
/*
```

COBOL applications on OS/390

Running COBOL programs on OS/390

The AMI needs access to the MQSeries datasets SCSQLOAD and SCSQAUTH, as well as one of the language-specific datasets such as SCSQANLE. See the MQSeries 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 (OS/390)" on page 489.

Chapter 10. 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 244
- "Reference information for the COBOL high-level interface" on page 246

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

AMHINIT (initialize) page 252 **AMHTERM** (terminate) page 267

Sending messages

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

AMHSNMS (send message) page 263 AMHSNRQ (send request) page 264 AMHSNRS (send response) page 265

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 256

AMHRCRQ (receive request) page 260

AMHBRMS (browse message)

page 249

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 262 AMHRCFL (receive file) page 254

Publish/subscribe

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

AMHPB (publish) page 253 AMHSB (subscribe) page 266 AMHUN (unsubscribe) page 268 AMHRCPB (receive publication)

page 258

Transaction support

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

AMHBEGIN (begin) page 248

COBOL high-level interface overview

page 251 AMHCMIT (commit) page 247 AMHBACK (backout)

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" on page 497).

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:

PIC S9(9) BINARY. 01 HSESSION 01 POLICY PIC X(n). PIC S9(9) BINARY. 01 COMPCODE 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).

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).

AMHBRMS (browse message)

Function to browse a message. See the MQSeries 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.
                 PIC X(n).
01 RECEIVER
01 POLICY
                 PIC X(n).
01 OPTIONS
                 PIC S9(9) BINARY.
01 BUFFLEN
                 PIC S9(9) BINARY.
01 DATALEN
                 PIC S9(9) BINARY.
                 PIC X(n).
01 DATA
                 PIC X(n).
01 RCVMSGNAME
   SENDER
                 PIC X(n).
01
                 PIC S9(9) BINARY.
01 COMPCODE
                 PIC S9(9) BINARY.
01 REASON
```

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)

 $\label{eq:ambrw-receive} \mbox{\sc AMRCRC (receive) for the} \\$

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 297). 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).

COBOL high-level interface

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).

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.

Reason code (output).

Declare the parameters as follows:

REASON

01 01 01 01 01	SESSNAME POLICY HSESSION COMPCODE REASON	PIC X(n). PIC X(n). PIC S9(9) BINARY. PIC S9(9) BINARY. 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.
HSE	SSION	The session handle (output).
COM	IPCODE	Completion 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		X(n).	
01	RESPNAME	PIC	X(n).	
01	TOPICLEN			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 297).

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).

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.
                PIC S9(9) BINARY.
01 REASON
```

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 297). 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:

	1			
01	HSESSION	PIC	S9(9)	BINARY.
01	RECEIVER	PIC	X(n).	
01	POLICY		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 297). 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

COBOL high-level interface

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.

I

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.
                PIC S9(9) BINARY.
01 BUFFLEN
                PIC S9(9) BINARY.
01 TOPICCOUNT
                PIC S9(9) BINARY.
01 TOPICLEN
01 FIRSTTOPIC
                PIC X(n).
                PIC S9(9) BINARY.
01 DATALEN
01 DATA
                PIC X(n).
                PIC X(n).
01 RCVMSGNAME
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 297).

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 297).

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 01	HSESSION RECEIVER		S9(9) X(n).	BINARY.
01	POLICY	PIC	X(n).	
01	BUFFLEN	PIC	S9(9)	BINARY.
01	DATALEN	PIC	S9(9)	BINARY.
01	DATA		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 297). 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).

COBOL high-level interface

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

ı

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			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 254 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 297). 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.
                PIC X(n).
01 SENDER
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 297).

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).

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.
                PIC X(n).
01 SENDER
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).
                PIC S9(9) BINARY.
01 COMPCODE
                PIC S9(9) BINARY.
01 REASON
```

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 297).

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).

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.
                PIC X(n).
01 SENDER
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).
                PIC S9(9) BINARY.
01 COMPCODE
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 297).

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).

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).
                 PIC X(n).
01 POLICY
01 RESPNAME
                PIC X(n).
                PIC S9(9) BINARY.
   TOPICLEN
01 TOPIC
                 PIC X(n).
01 FILTERLEN
                PIC S9(9) BINARY.
                PIC X(n).
01 FILTER
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 297).

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 MQSeries 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).

AMHTERM (terminate)

Closes the session, closes and deletes any implicitly created objects, and deletes the session. If MQSeries 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).

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		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 297).

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 MQSeries

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 11. 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:

page 270
page 272
page 274
page 275
page 276
page 277
page 278
page 279

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 280.

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 286
AMSEOP (open)	page 296
AMSECL (close)	page 285
AMSEDL (delete)	page 290

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 287

AMSECRSN (create sender) page 289

AMSECRRC (create receiver) page 288

AMSECRDL (create distribution list)

page 286

AMSECRPB (create publisher)

page 288

AMSECRSB (create subscriber)

page 289

AMSECRPO (create policy) page 287

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 293

AMSEGHSN (get sender handle)

page 295

AMSEGHRC (get receiver handle)

page 294

AMSEGHDL (get distribution list handle)

page 292

AMSEGHPB (get publisher handle)

page 294

AMSEGHSB (get subscriber handle)

page 295

AMSEGHPO (get policy handle)

COBOL object interface overview

Delete objects

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

AMSEDLMS (delete message)

page 290

AMSEDLSN (delete sender) page 292

AMSEDLRC (delete receiver) page 291

AMSEDLDL (delete distribution list)

page 290

AMSEDLPB (delete publisher)

page 291

AMSEDLSB (delete subscriber)

page 292

AMSEDLPO (delete policy) page 291

Transactional processing

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

AMSEBG (begin) page 284

AMSECM (commit) page 285

AMSERB (rollback) page 296

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 284

AMSEGTLE (get last error codes)

Message interface functions

A message object encapsulates an MQSeries 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 301

AMMSGTCI (get correl ID) page 302

AMMSGELC (get element CCSID)

page 301

AMMSGTEN (get encoding) page 304

AMMSGTFO (get format) page 305

AMMSGTGS (get group status)

page 306

AMMSGTMI (get message ID)

page 307

AMMSGTNA (get name) page 307

AMMSGTRC (get report code)

page 309

AMMSGTTY (get type) page 310

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 312

AMMSSTCI (set correl ID) page 312

AMMSSELC (set element CCSID)

page 313

AMMSSTEN (set encoding) page 313

AMMSSTFO (set format) page 314

AMMSSTGS (set group status)

page 314

Reset values

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

AMMSRS (reset) page 311

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)

COBOL object interface overview

AMMSGTDO (get data offset)

page 302

AMMSSTDO (set data offset) page 312

AMMSREBY (read bytes) page 311

AMMSWRBY (write bytes) page 315

Publish/subscribe topics

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

AMMSADTO (add topic) page 299

AMMSDETO (delete topic) page 301

AMMSGTTO (get topic) page 309

AMMSGTTC (get topic count)

page 310

Publish/subscribe filters

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

AMMSADFI (add filter) page 298

AMMSDEFI (delete filter) page 300

AMMSGTFI (get filter) page 305

AMMSGTFC(get filter count) page 304

Publish/subscribe name/value elements

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

AMMSADEL (add element) page 298

AMMSDEEL (delete element) page 299

AMMSGTEL (get element) page 303

AMMSGTEC (get element count)

page 303

AMMSDENE (delete named element)

page 300

AMMSGTNE (get named element)

page 308

AMMSGTNC (get named element count)

page 308

Error handling

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

AMMSCLEC (clear error codes)

page 299

AMMSGTLE (get last error) page 306

Sender interface functions

A sender object encapsulates an MQSeries object descriptor (MQOD) structure for sending a message.

Open and close

Functions to open and close the sender service.

AMSNOP (open) page 319 AMSNCL (close) page 317

Send

Function to send a message.

AMSNSN (send) page 319 AMSNSNFL(send file) page 320

Get values

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

AMSNGTCC (get CCSID) page 317 AMSNGTEN (get encoding) page 317 AMSNGTNA (get name) page 318

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 316

AMSNGTLE (get last error) page 318

Receiver interface functions

A receiver object encapsulates an MQSeries object descriptor (MQOD) structure for receiving a message.

Open and close

Functions to open and close the receiver service.

AMRCOP (open) page 328 AMRCCL (close) page 326

Receive and browse

Functions to receive or browse a message.

AMRCRC (receive) page 329

AMRCRCFL (receive file) page 330

AMRCBR (browse) page 322

AMRCBRSE (browse selection message) page 324

Get values

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

AMRCGTDT (get definition type)

page 326

AMRCGTNA (get name) page 327

AMRCGTQN (get queue name)

page 328

Set values

Function to set the queue name of the receiver service.

AMRCSTQN (set queue name)

page 331

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 325

AMRCGTLE (get last error) page 327

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 334 AMDLCL (close) page 332

Send

Function to send a message to the distribution list.

AMDLSN (send) page 335 AMDLSNFL (send file) page 335

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 333

AMDLGTSC (get sender count)

page 333

AMDLGTSH (get sender handle)

page 334

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 332

AMDLGTLE (get last error) page 332

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 339 AMPBCL (close) page 337

Publish

Function to publish a message.

AMPBPB (publish) page 340

Get values

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

AMPBGTCC (get CCSID) page 337 AMPBGTEN (get encoding) page 338 AMPBGTNA (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 publisher.

AMPBCLEC (clear error codes)

page 337

AMPBGTLE (get last error) page 338

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 345 AMSBCL (close) page 341

Broker messages

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

AMSBSB (subscribe) page 346 AMSBUN (unsubscribe) page 347 AMSBRC (receive) page 345

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 342

AMSBGTDT (get definition type)

page 342

AMSBGTEN (get encoding) page 343

AMSBGTNA (get name) page 344

AMSBGTQN (get queue name)

page 344

Set value

Function to set the queue name of the subscriber service.

AMSBSTQN (set queue name)

page 346

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 341

AMSBGTLE (get last error) page 343

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 349 AMPOGTWT (get wait time) page 349

Set value

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

AMPOSTWT (set wait time) page 350

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 348

AMPOGTLE (get last error) page 348

High-level functions

Each high-level function described in "Chapter 10. The COBOL high-level interface" on page 243 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

COBOL object interface overview

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 12. COBOL object interface reference

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

Session	page 284
Message	page 297
Sender	page 316
Receiver	page 322
Distribution list	page 332
Publisher	page 337
Subscriber	page 341
Policy	page 348

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" on page 497).

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 (MQSeries) connection object encapsulates a single MQSeries 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 MQSeries 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 MQSeries. The unit of work can subsequently be committed by AMSECM, or backed out by AMSERB. It should be used only when MQSeries 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.
   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.
                 PIC S9(9) BINARY.
01 COMPCODE
01 REASON
                 PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

COMPCODE Completion code (output). Reason code (output). REASON

AMSECL (close)

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

```
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).

COBOL session interface

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).

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).

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).

COBOL session interface

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).

AMSEDLMS (delete message)

REASON

Deletes a message object.

```
CALL 'AMSEDLMS' USING HSESS, HMSG, COMPCODE, REASON.
```

Reason code (output).

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).

The message handle returned by AMSECRMS (input).

COMPCODE Completion 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).

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).

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 an MQSeries 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).

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).

COBOL session interface

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 OS/390, programs running on OS/390 must use a local queue manager). The connection to the underlying message transport (MQSeries) 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).

Message interface functions

A *message* object encapsulates an MQSeries 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 237 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 311) and rebuild it each time.

Note that the following calls are only valid 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 298

AMMSDEEL (delete element) page 299

AMMSGTEL (get element) page 303

AMMSGTEC (get element count)

page 303

AMMSDENE (delete named element)

page 300

AMMSGTNE (get named element)

page 308

AMMSGTNC (get named element count)

page 308

AMMSADTO (add topic) page 299

AMMSDETO (delete topic) page 301

AMMSGTTO (get topic) page 309

AMMSGTTC (get topic count)

COBOL message interface

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 OPTIONS PIC S9(9) BINARY. 01 COMPCODE PIC S9(9) BINARY. 01 REASON PIC S9(9) BINARY.

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 MQSeries Integrator Version 2.0 Programming Guide.

COMPCODE Completion 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.
```

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			BINARY.
01	ELEMINDEX	PIC	S9(9)	BINARY.
01	COMPCODE	PIC	S9(9)	BINARY.
01	REASON	PIC	S9(9)	BINARY.

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).

COBOL message interface

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.
```

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.
```

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).

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.

The message handle returned by AMSECRMS (input).

CCSID The coded character set identifier (output).

COMPCODE Completion code (output).

COBOL message interface

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).
                PIC S9(9) BINARY.
01 COMPCODE
                PIC S9(9) BINARY.
01 REASON
```

HMSG The message handle returned by AMSECRMS (input).

The length in bytes of a buffer in which the correlation identifier is **BUFFLEN**

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.
   COMPCODE
01
                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:

```
PIC S9(9) BINARY.
01 HMSG
01 OFFSET
                 PIC S9(9) BINARY.
                 PIC S9(9) BINARY.
01 COMPCODE
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).

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.

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.

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).

COBOL message interface

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.
```

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.
```

The message handle returned by AMSECRMS (input).

COUNT The number of filters (output).

COMPCODE Completion 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.
```

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).

COBOL message interface

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 an MQSeries 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.
```

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.

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).

COBOL message interface

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.

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.
```

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).

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.
```

The message handle returned by AMSECRMS (input).

REPORTCODE The feedback code (output). The following values can be returned:

AMFB-EXPIRATION AMFB-COA AMFB-COD AMFB-ERROR

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.
```

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).

COBOL message interface

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.
```

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.
```

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).

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 312). 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			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 297).

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.
```

The message handle returned by AMSECRMS (input).

OPTIONS Reserved, must be specified as zero (input).

COMPCODE Completion code (output).

COBOL message interface

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:

```
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).

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 311 and "AMMSWRBY (write bytes)" on page 315.

```
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.
                 PIC S9(9) BINARY.
01
   REASON
```

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).

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.
```

The message handle returned by AMSECRMS (input).

ELEMENTCCSID The element coded character set identifier (input).

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).

COBOL message interface

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.
             PIC X(n).
PIC S9(9)
01 FORMAT
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.
                 PIC S9(9) BINARY.
01 COMPCODE
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).

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 312.

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.
```

The message handle returned by AMSECRMS (input).

WRITELEN The number of bytes to be written (input).

The data bytes (input).

COMPCODE Completion code (output).

Sender interface functions

A sender object encapsulates an MQSeries object descriptor (MQOD) structure. This represents an MQSeries 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 and policies" on page 469). 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. PIC S9(9) BINARY. 01 REASON

HSENDER The sender handle returned by AMSECRSN (input).

COMPCODE Completion code (output). **REASON** Reason code (output).

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).

The encoding (output).

COMPCODE Completion 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 an MQSeries 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).

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).

COBOL sender interface

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 330 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),

COBOL sender interface

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 an MQSeries object descriptor (MQOD) structure. This represents a local MQSeries 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 MQSeries 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.
                PIC S9(9) BINARY.
01 BUFFLEN
01 DATALEN
                PIC S9(9) BINARY.
01 DATA
                PIC X(n).
              PIC S9(9) BINARY.
01 HRCVMSG
              PIC S9(9) BINARY.
01 HSENDER
01 COMPCODE
                PIC S9(9) BINARY.
                PIC S9(9) BINARY.
01 REASON
```

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-NEXT) AMBRW-DEFAULT

COBOL receiver interface

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

ı

I

ı

ı

ı

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 *MQSeries 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.
                 PIC S9(9) BINARY.
01 HPOLTCY
01 OPTIONS
                 PIC S9(9) BINARY.
                 PIC S9(9) BINARY.
01 HSELMSG
01 BUFFLEN
                 PIC S9(9) BINARY.
                 PIC S9(9) BINARY.
01 DATALEN
01
   DATA
                 PIC X(n).
                 PIC S9(9) BINARY.
01
   HRCVMSG
01
   HRESPONSE
                 PIC S9(9) BINARY.
   COMPCODE
                 PIC S9(9) BINARY.
01
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 301)

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).

HRCYMSG 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

324

COBOL receiver interface

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

1

1

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).

COBOL receiver interface

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).

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 an MQSeries 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).

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).

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

I

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

COBOL receiver interface

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

COBOL receiver interface

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 312) 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 'AMRCSTON' 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.
              PIC X(II).
PIC S9(9) BINARY.
01 ERRORTEXT
01 REASON2
01 COMPCODE
                PIC S9(9) BINARY.
                 PIC S9(9) BINARY.
01 REASON
```

HDISTLIST The distribution list handle returned by AMSECRDL (input).

BUFFLEN Reserved, must be zero (input).

STRINGLEN Reserved (output). **ERRORTEXT** Reserved (output).

COBOL distribution list interface

REASON2 A secondary reason code (output). If REASON indicates

AMRC-TRANSPORT-WARNING or AMRC-TRANSPORT-ERR,

REASON2 gives an MQSeries 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).

COBOL distribution list interface

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.
                PIC S9(9) BINARY.
01 REASON
```

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.
                 PIC S9(9) BINARY.
01 HPOLICY
                 PIC S9(9) BINARY.
01 COMPCODE
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).

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).
                 PIC S9(9) BINARY.
01 HMSG
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).

COBOL distribution list interface

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 330 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).

COBOL publisher interface

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).

The encoding (output).

COMPCODE Completion code (output).

Peacon and (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 an MQSeries 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).

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).

COBOL publisher interface

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			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).

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).

COBOL subscriber interface

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.
               PIC S9(9) BINARY.
01 TYPE
01 COMPCODE
                PIC S9(9) BINARY.
                PIC S9(9) BINARY.
01 REASON
```

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

Completion code (output). COMPCODE

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 an MQSeries 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.

COBOL subscriber interface

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).

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 297).

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	DIC	(0)02	BINARY.
ÜΙ			. ,	
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).

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 297) 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.

COBOL subscriber interface

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 297) 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).

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 an MQSeries 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).

COBOL policy interface

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).

Part 5. The Java interface

Chapter 13. Using the Application Messaging		Send file
Interface in Java		Get values
Structure of the AMI	355	Error handling
Base classes	355	AmReceiver
Interface and helper classes	356	Open and close
Exception classes	356	Receive and browse
Using the repository	356	Receive file
System default objects	356	Get values
Writing applications in Java		Set value
Creating and opening objects		Error handling
Sending messages		AmDistributionList
Sample program		Open and close
Receiving messages		Send
Sample program		Send file
Request/response messaging		Get values
Sample programs		Error handling
File transfer	361	AmPublisher
Publish/subscribe messaging		Open and close
Sample programs		Publish
Using AmElement objects		Get values
Error handling		Error handling
Transaction support		AmSubscriber
Sending group messages		Open and close
Other considerations	365	Broker messages
Multithreading		Get values
Using MQSeries with the AMI		Set value
Field limits		Error handling
Building Java applications		AmPolicy
AMI package for Java		Policy management
		Error handling
Running Java programs	300	Helper classes
Observant A. James Indonésia a consensione	000	AmConstants
Chapter 14. Java interface overview		
Base classes		AmElement
Helper classes		AmObject
Exception classes	369	AmStatus
AmSessionFactory		Exception classes
Constructor		AmException
Session factory management		AmErrorException
Create session		AmWarningException
AmSession		
Session management		Chapter 15. Java interface reference
Create objects		Base classes
r	371	Helper classes
Error handling		Exception classes
AmMessage	372	AmSessionFactory
	372	AmSessionFactory
	372	createSession
	372	getFactoryName
Read and write data	372	getLocalHost
Publish/subscribe filters	372	getRepository
Publish/subscribe topics	373	getTraceLevel
	373	getTraceLocation
	373	setLocalHost
AmSender	374	setRepository
Open and close	374	setTraceLevel
Send		setTraceLocation

AmSession	enableWarnings
begin	getCCSID
clearErrorCodes	getEncoding
close	getLastErrorStatus
commit	getName
createDistributionList	open
createMessage	send
createPolicy	sendFile
createPublisher	
createReceiver	browse
createSender	clearErrorCodes
createSubscriber	close
enableWarnings	enableWarnings 400
getLastErrorStatus	getDefinitionType 400
getName	getLastErrorStatus 401
getTraceLevel	getName
getTraceLocation	getQueueName 401
open	open
rollback	receive
AmMessage	receiveFile
	setQueueName
addFilter	AmDistributionList
addTopic	clearErrorCodes 403
clearErrorCodes	close
deleteElement	enableWarnings 403
deleteFilter	getLastErrorStatus 403
deleteNamedElement	getName
deleteTopic	getSender
enableWarnings	getSenderCount
getCCSID	open
getCorrelationId	send
getDataLength	sendFile
getDataOffset	
getElement	clearErrorCodes
getElementCount	close
getEncoding	enableWarnings 405
getFilter	getCCSID
getFilterCount	getEncoding
getFormat	getLastErrorStatus 405
getGroupStatus	getName
getLastErrorStatus	open
getMessageId	publish
getName	AmSubscriber
getNamedElement	clearErrorCodes
getNamedElementCount	close
getReportCode	enableWarnings
	O
getTopic	getCCSID
getTopicCount	getDefinitionType
getType	getEncoding
readBytes	getLastErrorStatus
reset	getName
setCCSID	getQueueName 408
setCorrelationId	open
setDataOffset	receive
setEncoding	setQueueName
setFormat	subscribe
setGroupStatus	unsubscribe
writeBytes	AmPolicy
AmSender	•
clearErrorCodes	enableWarnings
close	getLastErrorStatus 410

getName							410
getWaitTime							
setWaitTime							410
AmConstants							413
AmElement							412
AmElement							412
AmConstants AmElement AmElement getName							412
getValue							412
getVersion							412
setVersion							412
toString							412
AmObject							413
clearErrorCodes							413
getLastErrorStatus.							413
getName							413
getName							414
AmStatus							414
getCompletionCod	e						414
getReasonCode.							414
getReasonCode2							414
toString							414
toString AmException getClassName							415
getClassName							415
getCompletionCod	e						415
getMethodName .							415
getReasonCode.							415
getSource toString							415
toString							415
AmErrorException							416
getClassName							416
getCompletionCod	e						416
getMethodName .							416
getReasonCode.							416
getMethodName getReasonCode . getSource toString							416
toString							416
AmWarningException							417
getClassName							417
getCompletionCode	e						417
getMethodName .							417
getClassName . getCompletionCod getMethodName getReasonCode .							417
getSource							417

Chapter 13. 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 357
- "Building Java applications" on page 366

Note that the term *object* is used in this book in the object-oriented programming sense, not in the sense of MQSeries '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 an MQSeries queue) to which messages are sent.

AmReceiver This is a service that represents a source (such as

an MQSeries 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.

Structure of the AMI

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

This is the base Exception class for amJava; all AmException

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 470).

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 470).

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 372 .

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 394) and rebuild it each time.

Sample program

For more details, refer to the SendAndForget.java sample program (see "Sample programs for AS/400, UNIX, and Windows" on page 461).

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 AS/400, UNIX, and Windows" on page 461).

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 357, 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 359, 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 359).

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 AS/400, UNIX, and Windows" on page 461).

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.

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 MQSeries Integrator Version 2" on page 458).

Sample programs

For more details, refer to the Publisher.java and Subscriber.java sample programs (see "Sample programs for AS/400, UNIX, and Windows" on page 461).

Using AmElement objects

Publish/subscribe brokers (such as MQSeries 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 MQSeries 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.

MQSeries 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.

- MQSeries 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 26.

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 (MQSeries) 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 MQSeries and a transport error (AMRC_TRANSPORT_ERR) will be reported to the application.

Using MQSeries with the AMI

You must not mix MQSeries 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 MQSeries, the underlying message transport. See the rules for naming MQSeries objects in the MQSeries 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 (AS/400)
/java/lib (UNIX)
\java\lib (Windows)

See "Directory structure" on page 430 (AIX), page 433 (AS/400), page 438 (HP-UX), page 445 (Solaris), or page 448 (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 429 (AIX), on page 432 (AS/400), page 437 (HP-UX), page 444 (Solaris), or page 447 (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, AS/400, HP-UX, Sun Solaris, Windows 98 and Windows NT 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 "Language compilers" on page 426 for versions of the Java Developer's Kit (JDK) supported by the AMI.

AIX Make sure that the JNI library libamtJava.so is accessible to your

runtime environment. To do this, you should perform:

export LIBPATH=\$LIBPATH:/usr/mqm/lib:

AS/400 Make sure that the library QMQMAMI is in the library list.

If you use the AS/400 QShell interpreter, you must use the export -s command, so that AMI can access the required

environment variables.

HP-UX Make sure that the JNI library libamtJava.sl is accessible to your

runtime environment. To do this, you should perform:

export SHLIB_PATH=\$SHLIB_PATH:/opt/mqm/lib:

|

Building Java applications

Make sure that the JNI library libamtJava.so is accessible to your **Solaris**

runtime environment. To do this, you should perform: 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 MQSeries 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

Building Java applications

Chapter 14. 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 370
AmSession	page 371
AmMessage	page 372
AmSender	page 374
AmReceiver	page 375
AmDistributionList	page 376
AmPublisher	page 377
AmSubscriber	page 378
AmPolicy	page 379

Helper classes

AmConstants	page 380
AmElement	page 380
AmObject	page 380
AmStatus	page 380

Exception classes

AmException	page 381
AmErrorException	page 381
AmWarningExcpetion	page 381

AmSessionFactory

The AmSessionFactory class is used to create AmSession objects.

Constructor

Constructor for AmSessionFactory.

AmSessionFactory page 384

Session factory management

Methods to return the name of an AmSessionFactory object, and to control traces.

getFactoryName	page 384
getLocalHost	page 384
getRepository	page 384
getTraceLevel	page 384
getTraceLocation	page 384
setLocalHost	page 384
setRepository	page 385
setTraceLevel	page 385
setTraceLocation	page 385

Create session

Method to create an AmSession object.

createSession page 384

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

open	page 388
close	page 386
getName	page 388
getTraceLevel	page 388
getTraceLocation	page 388

Create objects

Methods to create AmMessage, AmSender, AmReceiver, AmDistributionList AmPublisher, AmSubscriber, and AmPolicy objects.

createMessage	page 387
createSender	page 387
createReceiver	page 387
create Distribution List	page 386
createPublisher	page 387
createSubscriber	page 387
createPolicy	page 387

Transactional processing

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

begin	page 386
commit	page 386
rollback	page 388

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 386
enableWarnings	page 388
getLastErrorStatus	page 388

AmMessage

An **AmMessage** object encapsulates an MQSeries 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 391
getCorrelationId	page 391
getEncoding	page 392
getFormat	page 392
getGroupStatus	page 392
getMessageId	page 393
getName	page 393
getReportCode	page 393
getType	page 394

Set values

Methods to set the coded character set ID, correlation ID, format and group status of the message object.

setCCSID	page 394
setCorrelationId	page 394
setEncoding	page 395
setFormat	page 395
setGroupStatus	page 395

Reset values

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

reset page 394

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 391
getDataOffset	page 391
setDataOffset	page 394
readBytes	page 394
writeBytes	page 395

Publish/subscribe filters

Methods to manipulate filters for content-based publish/subscribe.

Java interface overview

addFilter	page 390
deleteFilter	page 390
getFilter	page 392
getFilterCount	page 392

Publish/subscribe topics

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

addTopic page 390 deleteTopic page 391 getTopic page 393 getTopicCount page 393

Publish/subscribe name/value elements

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

addElement page 389 deleteElement page 390 getElement page 391 getElementCount page 391 deleteNamedElement page 390 getNamedElement page 393 get Named Element Countpage 393

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 390 enableWarnings page 391 getLastErrorStatus page 392

AmSender

An **AmSender** object encapsulates an MQSeries object descriptor (MQOD) structure.

Open and close

Methods to open and close the sender service.

page 397 open close page 396

Send

Method to send a message.

send page 397

Send file

Method to send data from a file

sendFile page 398

Get values

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

getCCSID page 396 getEncoding page 396 getName page 397

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 396 enableWarnings page 396 getLastErrorStatus page 397

AmReceiver

An **AmReceiver** object encapsulates an MQSeries object descriptor (MQOD) structure.

Open and close

Methods to open and close the receiver service.

open page 401close page 400

Receive and browse

Methods to receive or browse a message.

receive page 401

browse page 399

Receive file

Method to receive file message data into a file.

receiveFile page 402

Get values

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

getDefinitionTypepage 400getNamepage 401getQueueNamepage 401

Set value

Method to set the queue name of the receiver service.

setQueueName page 402

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodespage 400enableWarningspage 400getLastErrorStatuspage 401

AmDistributionList

An AmDistributionList object encapsulates a list of AmSender objects.

Open and close

Methods to open and close the distribution list service.

open page 403 close page 403

Send

Method to send a message to the distribution list.

send page 404

Send file

Method to send date from a file to each sender defined in the distribution list.

sendFile page 404

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 403 getSenderCount page 403 getSender page 403

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 403 enableWarnings page 403 getLastErrorStatus page 403

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.

page 406 open close page 405

Publish

Method to publish a message.

publish page 406

Get values

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

getCCSID page 405 getEncoding page 405 getName page 405

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 405 enableWarnings page 405 getLastErrorStatuspage 405

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.

page 408 open close page 407

Broker messages

Methods to subscribe to a broker, remove a subscription, and receive a publication from the broker.

subscribe page 409 unsubscribe page 409 receive page 408

Get values

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

getCCSID page 407 getDefinitionType page 407 getEncoding page 407 getName page 408 getQueueName page 408

Set value

Method to set the queue name of the subscriber service.

setQueueName page 408

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 407 enableWarnings page 407 getLastErrorStatus page 407

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.

getNamepage 410getWaitTimepage 410setWaitTimepage 410

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodespage 410enableWarningspage 410getLastErrorStatuspage 410

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 411

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 412
getName	page 412
getValue	page 412
getVersion	page 412
setVersion	page 412
toString	page 412

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 413
getLastErrorStatus	page 413
getName	page 413

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 414
get Completion Code	page 414
getReasonCode	page 414
getReasonCode2	page 414
toString	page 414

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 415
get Completion Code	page 415
get Method Name	page 415
getReasonCode	page 415
getSource	page 415
toString	page 415

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 416
get Completion Code	page 416
getMethodName	page 416
getReasonCode	page 416
getSource	page 416
toString	page 416

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 417
get Completion Code	page 417
getMethodName	page 417
getReasonCode	page 417
getSource	page 417
toString	page 417

Java interface overview

Chapter 15. 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 411.

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.

AmSessionFactory	page 384
AmSession	page 386
AmMessage	page 389
AmSender	page 396
AmReceiver	page 399
AmDistributionList	page 403
AmPublisher	page 405
AmSubscriber	page 407
AmPolicy	page 410

Helper classes

AmConstants	page 411
AmElement	page 412
AmObject	page 413
AmStatus	page 414

Exception classes

AmException	page 415
AmErrorException	page 416
AmWarningException	page 417

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 (AS/400, UNIX, and Windows)" on page 450 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

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.

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 (MQSeries) connection object encapsulates a single MQSeries 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 MQSeries. 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);

warnings0n 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 MQSeries and the MQSeries local queue manager library cannot be loaded, the MQSeries 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 MQSeries 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 394) and rebuild it each time.

Note that the following methods are only valid after a session has been opened with **AmSession.open**:

page 389
page 390
page 391
page 391
page 390
page 393
page 393
page 390
page 391
page 393
page 393

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.

Java AmMessage

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 <code>getLastErrorStatus</code>, even if AmWarningExceptions are disabled.

void enableWarnings(boolean warningsOn);

warnings0n 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();
```

Java AmMessage

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.

getReportCode

Returns the feedback code from an AmMessage of type MQMT_REPORT.

```
int getReportCode();
```

The following values can be returned:

```
AMFB_NONE
AMFB_EXPIRATION
AMFB_COA
AMFB_COD
AMFB_ERROR
```

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();

Java AmMessage

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 389).

```
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

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 an MQSeries object descriptor (MQOD) structure. This represents an MQSeries 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);

warnings0n 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.

Java AmSender

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);

The message object to use to send the file. This can be used to sendMessage

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 402 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 an MQSeries object descriptor (MQOD) structure. This represents an MQSeries 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 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 394) 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);

warnings0n 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.

400

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 394) 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 394) 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);
```

warnings0n 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.

Java AmDistributionList

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);

The message object containing the data to be sent. sendMessage

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, fileName, String 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 402 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);

warnings0n 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();
```

Java AmPublisher

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);

warnings0n 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();
```

Java AmSubscriber

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 389).

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 389) 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 389) 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);

warnings0n 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" on page 509.

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. The value of the element. value

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 MQSeries, the secondary reason code is an MQSeries 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();

Java AmWarningException

Part 6. OS/390 Subsystems

Chapter 16. Writing applications for OS/390	
subsystems	. 421
Writing IMS applications using AMI	. 421
Writing CICS applications using AMI	. 421
Writing batch applications using AMI	. 422
Writing RRS-batch applications using AMI .	. 422
RRS availability	. 422

Chapter 16. Writing applications for OS/390 subsystems

Here is some advice for those of you who want to write AMI applications for the IMS, CICS, batch, and RRS-batch subsystems on OS/390.

Writing IMS applications using AMI

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 MQSeries 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.

We recommend 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.

Writing CICS applications using AMI

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 OS/390, 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 (OS/390)" on page 492 for more details.

Writing batch applications using AMI

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 MQSeries and DB2®, within a single unit of work, you should use RRS. For further information, see "Writing RRS-batch applications using AMI".

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.

Writing RRS-batch applications using AMI

Transaction management and recoverable resource services (RRS) is an OS/390 facility that provides two-phase syncpoint support across participating resource managers. An application can update recoverable resources managed by various OS/390 resource managers such as MQSeries 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 OS/390 system, any AMI call which resolves to an MQSeries 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.

Part 7. Setting up an AMI installation

Chapter 17. Installation and sample programs	425	Local host and repository files (AS/400, UNIX, and
Prerequisites	. 425	Windows) 45
Disk space	. 425	Default location 45
Operating environments	. 425	Default names 45
MQSeries environment	. 426	Overriding the default location and names 45
Language compilers		Local host file 45
Installation on AIX		Repository file
Installation	. 428	Local host and repository files (OS/390) 45
Manual installation	. 428	Batch, RRS-batch, IMS 45
Using amtInstall	. 428	CICS
Removing the AMI		Local host file
Setting the runtime environment		Repository file 45
Java programs	. 429	Repository and local host caches 45
Directory structure (AIX)		Generating caches 45
Installation on AS/400		Using a cache 45
Setting the runtime environment for Java		Cache generator messages 45
programs	. 432	The administration tool 45
Directory structure (AS/400)	. 433	Installation 45
Library QMQMAMI	. 433	Operation
Members of QMQMAMI/H		Connecting to MQSeries
Members of QMQMAMI/AMTMQSC		Using MQSeries Integrator Version 1 45
Members of QMQMAMI/QCSRC		Using MQSeries Publish/Subscribe 45
Members of QMQMAMI/QCPPSRC		Using MQSeries Integrator Version 2 45
IFS		Migrating to MQSeries Integrator V2 from V1
Installation on HP-UX		and MQSeries Publish/Subscribe 45
Installation		Creating default MQSeries objects 46
Manual installation		The sample programs 46
Using amtInstall		Sample programs for AS/400, UNIX, and
Removing the AMI	. 436	Windows
Setting the runtime environment		Running the AS/400, UNIX, and Windows
Java programs	. 437	sample programs 46
Directory structure (HP-UX)		MQSeries objects
Installation on OS/390		Repository and host files 46
Installation		Running the publish/subscribe samples 46
Setting the runtime environment		Setting the runtime environment 46
Batch and RRS-batch		Running the C and C++ samples 46
IMS		Running the Java samples 46
CICS		Running the sample programs (AS/400) 46
Unicode character conversion		Sample programs for OS/390
Batch, RRS-batch, IMS		Running the sample programs (OS/390) 46
CICS		Building the sample programs 46
Directory structure (OS/390)	. 441	MQSeries objects
Installation on Sun Solaris		Repository and host files
Installation		Running the publish/subscribe samples 46
Manual installation		Setting the runtime environment 46
Using amtInstall		File name input for the file transfer samples 46
Removing the AMI		Running the batch samples
Setting the runtime environment		Running the CICS samples
Java programs		Running the IMS samples
Directory structure (Solaris)		- and the first samples
Installation on Windows.		Chapter 18. Defining services and policies 46
Installation		Services and policies
Removing the AMI		System provided definitions
Setting the runtime environment		System default objects
Directory structure (Windows)		Service definitions
Directory structure (vindows)	. 110	Service point (sender/receiver)
		bervice point (sender/receiver) 4/

Distribution list							. 474
Subscriber							
Publisher							. 474
Policy definitions							. 475
Initialization attributes							. 475
General attributes							. 476
Send attributes							
Receive attributes							
Subscribe attributes							
Publish attributes							. 480
Chapter 19. Problem determina	atio	n					. 481
Using trace (AS/400, UNIX, and	W	ind	low	7S)			. 481
Trace filename and directory							. 482
Commands on AS/400 .							
Commands on UNIX							. 482
Commands on Windows							. 483
C++ and Java							. 484
Example trace							. 484
Using trace (OS/390)							. 489
Formatted Trace							. 489
Control of formatted trace .							. 489
GTF Trace							. 490
Control of GTF Trace							. 490
Control of GTF Trace When your AMI program fails							. 492
Reason Codes							. 492
First failure symptom report	(AS	/4	00,	UN	ΛIN	ζ,	
First failure symptom report and Windows)							. 492
First failure symptom report	(OS	/3	90)				. 492
Other sources of information							
Common causes of problems							

Chapter 17. Installation and sample programs

The Application Messaging Interface is available for the AIX, AS/400, HP-UX, OS/390, Sun Solaris, Windows NT, and Windows 98 platforms.

This chapter contains:

- · "Prerequisites"
- "Installation on AIX" on page 428
- "Installation on AS/400" on page 432
- "Installation on HP-UX" on page 436
- "Installation on OS/390" on page 440
- "Installation on Sun Solaris" on page 443
- "Installation on Windows" on page 447
- "Local host and repository files (AS/400, UNIX, and Windows)" on page 450
- "The administration tool" on page 457
- "Connecting to MQSeries" on page 458
- "The sample programs" on page 461

Prerequisites

I

Before you install the AMI, ensure that your system has sufficient disk space, and has the software listed in the following sections.

Disk space

Disk space requirements:

AIX 15.0 MB AS/400 30.0 MB HP-UX 12.7 MB

OS/390 Not applicable (AMI installed as part of MQSeries

for OS/390)

Sun Solaris 11.2 MB

Windows 10.9 MB (without AMI Administration Tool)

22.4 MB (with AMI Administration Tool)

Operating environments

The AMI runs under the following operating systems:

AIX V4.3

AS/400 V4R4 or later

HP-UX V11.0

OS/390 V2R6 or later, with Language Environment

CICS 4.1 or later, with Language Environment

IMS V5.1 or later, with Language Environment

Sun Solaris V2.6 and V7

Windows NT V4 and Windows 98

Prerequisites

MQSeries environment

You can run the AMI in an MQSeries server or client environment.

To run the AMI in an MQSeries server environment, you need at least one of the following installed on your system:

- MQSeries for AIX Version 5.1 or later
- MQSeries for AS/400 Version 5.1 or later
- MQSeries for HP-UX Version 5.1 or later
- MQSeries for OS/390 Version 5.2
- MQSeries for Sun Solaris Version 5.1 or later
- MQSeries for Windows NT Version 5.1 or later

To run the AMI in an MQSeries client environment, you need at least one of the following installed on your system:

- MQSeries client for AIX Version 5.1 or later
- MQSeries client for HP-UX Version 5.1 or later
- MQSeries client for Sun Solaris Version 5.1 or later
- MQSeries client for Windows NT Version 5.1 or later
- MQSeries client for Windows 98 Version 5.1 or later

The MQSeries client requires access to at least one supporting MQSeries server.

Language compilers

The following language compilers for C, COBOL, C++, and Java are supported:

VisualAge® C++ 5.0 **AIX**

JDK[™] 1.1.7 and later

AS/400 AS/400 Developer Kit for Java (5769JV1)

> ILE C for AS/400 (5769CX2) ILE C++ for AS/400 (5799GDW) VisualAge C++ for OS/400 (5716CX4)

HP-UX HP aC++ B3910B A.03.10

HP aC++ B3910B A.03.04 (970930) Support library

JDK 1.1.7 and later

OS/390 OS/390 C/C++ Version 2 Release 6 and later

IBM COBOL for OS/390 & VM Version 2 Release 1

and later

IBM COBOL for MVS & VM Version 1 Release 2

and later

Sun Solaris Workshop Compiler 4.2 (with Solaris 2.6)

Workshop Compiler 5.0 (with Solaris 7)

JDK 1.1.7 and later

Windows Microsoft Visual C++ 6

JDK 1.1.7 and later

Next step

Now go to one of the following to start the installation procedure:

- "Installation on AIX" on page 428
- "Installation on AS/400" on page 432
- "Installation on HP-UX" on page 436
- "Installation on OS/390" on page 440
- "Installation on Sun Solaris" on page 443
- "Installation on Windows" on page 447

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 MQSeries directory /usr/mqm, so that the AMI tar file restores to a directory structure consistent with MQSeries. This operation usually requires root access. Existing files will be overwritten. (Note that the location /usr/mqm/ is consistent with MQSeries Version 5.1, which is the prerequisite for the AMI).

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 430.

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:

 ${\tt export\ PATH=\$PATH:/usr/mqm/amt/samp/C/bin:/usr/mqm/amt/samp/Cpp/bin:}$

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:

Also, to load the AMI library for Java:

export LIBPATH=\$LIBPATH:/usr/mqm/lib:

Next step

Now go to "Local host and repository files (AS/400, UNIX, and Windows)" on page 450 to continue the installation procedure.

Directory structure (AIX)

The AMI tar file contains:

```
/amt/amtsdfts.tst : MQSeries mgsc command file to create default MQSeries
 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/inc
   amtc.h : The C header file for the AMI
   amtcpp.hpp: The C++ header file for the AMI
  oamasami.h: The C header file for the OAMAS AMI subset
/amt/ipla : The International Program License Agreement file
/amt/li : The License Information file
/iava/lib
   com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java
   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 MQSeries Server threaded library
   amtcmqic: Dynamic binding stub for MQSeries Client library
  amtcmqic r : Dynamic binding stub for MQSeries Client threaded library
/amt/samp
   amtsamp.tst : MQSeries mqsc command file to create MQSeries 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
```

Installation on AIX

```
amtsorcy: 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
   SendAndForget.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
   ReceiveFile.cpp : C++ source for receive file sample
  SendFile.cpp : C++ source for send file sample
/amt/samp/Cpp/bin
   SendAndForget : 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
   ReceiveFile : C++ source for receive file sample
  SendFile : C++ source for send file sample
/amt/samp/java
   SendAndForget.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
   ReceiveFile.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
```

I	Installation on AS/400
	The AMI package for AS/400 comes as a compressed zip file ma0f_400.zip. Uncompress and restore it as follows:
I	1. Download ma0f_400.zip to a directory on your PC.
I	2. Uncompress the file using the InfoZip Unzip program.
I	The file ma0f_400.sav is created.
 	3. Create a save file called MA0F in a suitable library on the AS/400, 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\MAOF_400.SAV QGPL/MAOF
I	5. Install the MQSeries for AS/400 AMI, product Id 5724A23, using RSTLICPGM:
l	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)
i	6. Delete the save file created in Step 3:
	DLTF FILE(QGPL/MA0F)
	To remove the AMI package from the AS/400, use DLTLICPGM: DLTLICPGM LICPGM(5724A23)
I	Setting the runtime environment for Java programs
	To run Java:
	 Ensure that the library QMQMAMI is in the library list.
l	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:
I	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:
I	:/QIBM/ProdData/mqm/amt/java/lib/com.ibm.mq.amt.jar
I	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.
ı	Next step
	Now go to "Local host and repository files (AS/400, UNIX, and Windows)" on page 450 to continue the installation procedure.

Directory structure (AS/400)

I

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
                       : The AMI JNI library
            *SRVPGM
AMTJAVA
            *SRVPGM
                      : The main AMI XML parsing library
AMTXML
AMTMSG
            *MSGF
                      : AMT message file
QAMT0050
            *PRDDFN
                      : Product definition file
            *PRDLOD
QAMT0029
                      : Language Product Load file
                      : Code Product Load file
QAMT0050
            *PRDLOD
                      : AMI header files
            *FILE
Н
                       : MQSC command files
AMTMQSC
            *FILE
QXMLMSG
            *MSGF
                       : XML message file
            *PGM
                       : C high-level client sample program
AMTSHCLT
            *PGM
                       : C high-level receive file sample program
AMTSHFRC.
                       : C high-level send file sample program
AMTSHFSN
            *PGM
AMTSHPUB
            *PGM
                       : C high-level subscriber sample program
AMTSHRCV
            *PGM
                       : C high-level receiver sample program
            *PGM
AMTSHSND
                       : C high-level send and forget sample program
AMTSHSUB
            *PGM
                       : C high-level subscriber sample program
            *PGM
                       : C high-level server sample program
AMTSHSVR
AMTSOCLT
            *PGM
                       : C object-level client sample program
            *PGM
                       : C object-level receive file sample program
AMTSOFRC
            *PGM
                       : C object-level send file sample program
AMTSOFSN
                       : C object-level publisher sample program
AMTSOPUB
            *PGM
AMTSORCV
            *PGM
                       : C object-level receiver sample program
            *PGM
AMTSOSGR
                       : C object-level receive group sample program
            *PGM
AMTSOSGS
                       : 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
            *PGM
CLIENT
                       : C++ client sample program
            *PGM
                      : C++ publisher sample program
PUBLISHER
RECEIVER
            *PGM
                      : C++ receiver sample program
            *PGM
                      : C++ receive file sample program
RCVFILE
SENDFORGET
           *PGM
                      : C++ send and forget sample program
            *PGM
                      : C++ send file sample program
SENDFILE
            *PGM
                      : C++ server sample program
SFRVFR
SUBSCRIBER *PGM
                      : C++ subscriber sample program
            *FILE
                       : C sample files
QCSRC
QCPPSRC
            *FILE
                       : C++ samples
AMTTOX1C
            *PGM
                       : Installation Exit Program
AMTI0X0C
            *PGM
                       : Installation Exit Program
```

Members of QMQMAMI/H

AMTC : The C header file for AMI
AMTCPP : The C++ header file for AMI

Note: The members of the file H are copies of the AMI header files in IFS.

Members of QMQMAMI/AMTMQSC

AMTSDFTS : MQSeries mqsc command file to create default MQSeries objects required by the AMI

AMTSAMP : MQSeries mqsc command file to create MQSeries objects required by AMI samples

Members of QMQMAMI/QCSRC

amtshclt
 c source for high-level client sample program
 mtshfrc
 c source for high-level receive file sample program
 c source for high-level send file sample program

Installation on AS/400

```
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
: 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
amtsorcv
amtsosgr
amtsosgs
                : C source for object-level publisher sample program
                : C source for object-level receiver sample program
                : C source for object-level receive group sample program : 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 : C source for object-level server sample program
                : C source for object-level subscriber sample program
```

Note: The members of the file QCSRC are copies of the C source files for sample programs 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
            : C++ source for server sample
Server
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.

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
              : AMI header files
    inc
                : The International Program License Agreement file
    ipla
    1i
                : The License Information file
/QIBM/ProdData/mqm/amt/inc
    amtc.h
               : The C header file for AMI
    amtcpp.hpp : The C++ header file for AMI
/QIBM/ProdData/mgm/amt/samp
    amtsamp.tst : MQSeries mqsc command file to create MQSeries 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
```

Installation on AS/400

```
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/Cpp
                    : C++ source for client sample
     Client.cpp
                        : C++ source for publisher sample
     Publisher.cpp
     Receiver.cpp
                        : C++ source for receiver sample
     RcvFile.cpp
                        : C++ source for receive file sample
     SendForget.cpp \,:\, C++ source for send and forget sample
                        : C++ source for send file sample
     SendFile.cpp
Server.cpp
                         : C++ source for server sample
     Subscriber.cpp
                        : C++ source for subscriber sample
/QIBM/ProdData/mgm/amt/samp/java
                      : Java source for client sample
     Client.java
     Publisher.java : Java source for publisher sample
Receiver.java : Java source for receiver sample
ReceiveFile.java : Java source for receive file sample
     SendandForget.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
 /QIBM/ProdData/mgm/amt/java/lib
     com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java
 /OIBM/UserData/mgm/amt
     amt.dtd: Document Type Defintion file on which the AMI repository is based
     amthost.xml
                         : Sample AMI XML file used as the default host file
```

Ι

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 MQSeries directory /opt/mqm, so that the AMI tar file restores to a directory structure consistent with MQSeries. 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 438.

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:

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 (AS/400, UNIX, and Windows)" on page 450 to continue the installation procedure.

Directory structure (HP-UX)

The AMI tar file contains:

```
/amt/amtsdfts.tst : MQSeries mgsc command file to create default MQSeries
 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/inc
   amtc.h : The C header file for the AMI
   amtcpp.hpp : The C++ header file for the AMI
  oamasami.h : The C header file for the OAMAS AMI subset
/amt/ipla : The International Program License Agreement file
/amt/li : The License Information file
/iava/lib
   com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java
   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.
  amtcmgm r : Dynamic binding stub for MQSeries Server threaded library
  amtcmqic r : Dynamic binding stub for MQSeries Client threaded library
/amt/samp
  amtsamp.tst : MQSeries mqsc command file to create MQSeries 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
  \verb|amtsofsn|: C object-level send file sample program|\\
```

Installation on HP-UX

```
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
  SendAndForget.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
   ReceiveFile.cpp : C++ source for receive file sample
  SendFile.cpp : C++ source for send file sample
/amt/samp/Cpp/bin
   SendAndForget : 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
   ReceiveFile : C++ source for receive file sample
  SendFile : C++ source for send file sample
/amt/samp/java
   SendAndForget.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
   ReceiveFile.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 OS/390

The AMI is installed automatically with MQSeries for OS/390 Version 5.2.

Installation

The files and directories created are listed in "Directory structure (OS/390)" on page 441.

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.

Unicode character conversion

If your OS/390 installation predates OS/390 V2 R9, applications that use the AMI publish subscribe calls, message element calls, and file transfer calls may need to perform some extra configuration. This configuration enables the Language Environment support for Unicode character conversion. With OS/390 V2 R9, the Unicode conversion tables were replaced with direct Unicode converters, enabling higher performance and removing the need for this extra configuration. Refer to the OS/390 V2R9.0 C/C++ Compiler and Run-Time Migration Guide for more details.

Batch, RRS-batch, IMS

If your Language Environment is installed in a non-default location, you will need to set the environment variable ICONV UCS2 PREFIX to specify the value of your installation prefix before running your AMI application. This ensures that the AMI has access to Unicode character conversion tables. See the OS/390 C/C++ Programming Guide for examples of setting this environment variable.

CICS

OS/390 releases before OS/390 V2 R9 do not support Unicode character conversions under CICS. This makes it impossible to use AMI publish subscribe and message element support with earlier versions of OS/390.

OS/390 V2 R9 is required to enable AMI publish subscribe or message element support under CICS.

Next step

Now go to "Local host and repository files (OS/390)" on page 453 to continue the installation procedure.

Directory structure (OS/390)

On OS/390 platforms the directory structure contains the following (where 'hlq' is the high-level qualifier of the AMI installation):

```
hla.SCSOLOAD
  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
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
hla.SCSOC370
  AMTC: The C header file for the AMI
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
hlg.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: MQSeries mgsc command file to create default MQSeries objects
    required by the AMI.
  AMTSAMP: MQSeries mqsc command file to create MQSeries 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)
hlg.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
```

Installation on OS/390

```
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
```

Installation on Sun Solaris

The AMI package for Sun 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 installationamtRemove 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 MQSeries directory /opt/mqm, so that the AMI tar file restores to a directory structure consistent with MQSeries. 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 445.

Removing the AMI

Run the amtRemove utility to remove all the files that were created by amtInstall.

Installation on Sun Solaris

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:

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 (AS/400, UNIX, and Windows)" on page 450 to continue the installation procedure.

Directory structure (Solaris)

The AMI tar file contains:

```
/amt/amtsdfts.tst : MQSeries mgsc command file to create default MQSeries
 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/inc
   amtc.h : The C header file for the AMI
   amtcpp.hpp : The C++ header file for the AMI
  oamasami.h : The C header file for the OAMAS AMI subset
/amt/ipla : The International Program License Agreement file
/amt/li : The License Information file
/iava/lib
   com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java
   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 MQSeries Server library
  amtcmqic : Dynamic binding stub for MQSeries Client library
/amt/samp
   amtsamp.tst : MQSeries mqsc command file to create MQSeries 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
```

Installation on Sun Solaris

```
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
   SendAndForget.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
   ReceiveFile.cpp : C++ source for receive file sample
   SendFile.cpp : C++ source for send file sample
/amt/samp/Cpp/bin
   SendAndForget : 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
   ReceiveFile : C++ source for receive file sample
  SendFile : C++ source for send file sample
/amt/samp/java
   SendAndForget.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
   ReceiveFile.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 and Windows NT 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 MQSeries 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 448.

Removing the AMI

To uninstall the Application Messaging Interface, use the Add/Remove Programs control panel.

Note: You **must** remove the AMI entries from the CLASSPATH (for instance,

C:\MQSeries\java\lib\com.ibm.mq.amt.jar; and

C:\MQSeries\amt\samples\java\bin\com.ibm.mq.amt.samples.jar;). These will not be removed by Add/Remove Programs.

In addition, if you specified a directory other than the default during installation, you must remove this directory from the PATH environment variable.

Setting the runtime environment

By default, the location of the AMI runtime binary files matches that of MQSeries (for example C:\MQSeries\bin). If you specified a different directory for the runtime files, you **must** add it to the PATH environment variable. (See also "Removing the AMI".)

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 MQSeries is C:\MQSeries):

```
set PATH=%PATH%;C:\MQSeries\amt\samples\C\bin;
    C:\MQSeries\amt\samples\Cpp\bin;
```

When running Java, the AMI classes (C:\MQSeries\java\lib\com.ibm.mq.amt.jar) and samples (C:\MQSeries\amt\samples\java\bin\com.ibm.mq.amt.samples.jar) must be contained in the CLASSPATH environment variable. This is done by the setup program.

Next step

Now go to "Local host and repository files (AS/400, UNIX, and Windows)" on page 450 to continue the installation procedure.

Directory structure (Windows)

On Windows platforms the directory structure contains:

```
\amt\amtsdfts.tst : MQSeries mgsc command file to create default MQSeries
 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\include
  amtc.h : The C header file for the AMI
  amtcpp.hpp: The C++ header file for the AMI
  oamasami.h: The C header file for the OAMAS AMI subset
\amt\ipla : The International Program License Agreement file
\amt\li : The License Information file
   com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java
\hin
  amt.dll: The main AMI library
  amt.lib : The AMI LIB file used for building C programs
  amtXML310.dll : The AMI XML parsing library
   amtCpp.dll : The AMI C++ library
   amtCpp.lib : The AMI LIB file used for building C++ programs
   amtJava.dll: The AMI JNI library
   amtICUUC140.dll : The AMI codepage translation library
  amtICUDATA.dll: The AMI codepage translation data library
  MSVCRT.DLL: Main MVSC runtime library
  MSVCIRT.DLL: Iostream MSVC runtime library
\amt\samples
  amtsamp.tst : MQSeries mqsc command file to create MQSeries 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
```

Installation on Windows

```
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
   SendAndForget.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
   ReceiveFile.cpp : C++ source for receive file sample
   SendFile.cpp : C++ source for send file sample
\amt\samples\Cpp\bin
   SendAndForget.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
   ReceiveFile.exe : C++ receive file sample program
   SendFile.exe : C++ send file sample program
\amt\samples\java
   SendAndForget.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
   ReceiveFile.java : Java source for receive file sample
   SendFile.java : Java source for send file sample
\amt\samples\java\bin
   com.ibm.mg.amt.samples.jar: The jar file containing the AMI
     samples class files for Java
```

Local host and repository files (AS/400, 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 AS/400, the default directory for the files is:

/QIBM/UserData/mgm/amt

On UNIX®, the default directory for the files is:

/usr/mgm/amt

/opt/mqm/amt (HP-UX, Solaris)

On Windows, the default location is a directory called \amt under the user specified MQSeries file directory. For example, if MQSeries is installed in the C:\MQSeries directory, the default directory for the AMI data files on Windows NT

C:\MQSeries\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 (AS/400)

(UNIX) /amt/samp

\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')
                                                    (AS/400)
```

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') (AS/400)
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

Local host and repository files (AS/400, UNIX, and Windows)

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 MQSeries 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.

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 = "OMNAME"
```

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.

Local host and repository files (AS/400, UNIX, and Windows)

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 provides definitions for policies and services. If you do not use a repository file, AMI uses its built-in definitions. For more information, see "Chapter 18. Defining services and policies" on page 469.

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 *OS/390 Language Environment for OS/390 and VM 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 OS/390 C/C++ function setenv():

```
setenv( "AMT REPOSITORY", "NAME", 1 );
```

Local host file

An AMI installation using OS/390 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 MQSeries queue manager that you want to connect to on your OS/390 installation. (The local host file is not needed for CICS, because there is only one MQSeries 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:

```
= "OMNAME1"
connectionName1
```

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" 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. For more information, see "Chapter 18. Defining services and policies" on page 469.

"Repository and local host caches" explains how to use a repository cache instead of a repository file.

Repository and local host caches

On OS/390, 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 OS/390 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.

Here is a sample JCL fragment to run the cache generator (with US English messages):

```
//GO EXEC PGM=AMTASM10
//STEPLIB DD DSN=h1q.SCSQLOAD,DISP=SHR
// DD DSN=h1q.SCSQANLE,DISP=SHR
//AMTHOST DD DSN=h1q.SCSQPROC(AMTHOST),DISP=SHR
//AMT DD DSN=h1q.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 returned by the cache generator are described in the following section.

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 453 and "CICS" on page 453.

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.

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 MQSeries 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 MQSeries Application Messaging Interface Manual for an */
/* explanation of CompCode, AMCC, and Reason, AMRC. */
"AMT0004I AMI repository cache created"
/* Explanation:
/* A repository cache was successfully created.
/* User Response:
/* None.
"AMT0005I AMI host file cache created"
/* Explanation:
/* A host file cache was successfully created.
/* User Response:
/* None.
```

The administration tool

The AMI administration tool is for use on Windows NT Version 4 only.

Installation

The 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 447). It is installed in sub-directory amt\AMITool.

To start the AMI administration program, select **IBM MQSeries AMI \ IBM MQSeries 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\AMItool directory, and open the file amiSample.xml. You should see a number of services and policies 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 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

The default attributes provided by the tool are as specified in "Service definitions" on page 472 and "Policy definitions" on page 475.

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 and policies 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 (AS/400, UNIX, and Windows)" on page 450). 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.

Note: 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.

Further information can be found in the AMI administration tool online help.

Connecting to MQSeries

You can connect to MQSeries, the transport layer, using an MQSeries server or an MQSeries 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 an MQSeries client and an MQSeries 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 475).

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 472). 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 475). 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 MQSeries). 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

If you want to use the publish/subscribe functions of the AMI, you must have MQSeries Publish/Subscribe installed (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 472). This causes an MQRFH header containing publish/subscribe name/value elements to be added to a message when it is sent.

Using MQSeries Integrator Version 2

You can use your existing AMI repository file, MQSeries Publish/Subscribe applications, and MOSeries Integrator Version 1 (MOSI V1) applications unchanged with MQSeries Integrator Version 2 (MQSI V2).

Alternatively, if you are writing a new application or wish to exploit some of the additional function provided by MQSI V2, you should specify 'MQSeries 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 MQSI V2 and to exploit its publish and subscribe functions.

Applications send messages to MQSI V2 using the standard AMI send verbs. If the service point has been defined as a Service Type of 'MQSeries Integrator V2', the AMI will automatically build an MQRFH2 header at the beginning of the message and add 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 MQSI V2. Applications requiring more control can explicitly add the MCD information using the amMsgAddElement C, AMSADEL COBOL, or

Connecting to MQSeries

AmMessage::addElement C++ and Java calls. The default MCD values will be ignored if the application has added the elements to the message explicitly. The MQRFH2 and MCD fields are described in the *MQSeries 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 MQSeries Integrator Version 2 Programming Guide.

If you specify the Service Type as 'RF Header V2', the AMI will select and use the Publish and Subscribe policy options applicable to MQSI 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 'MQSeries Integrator V2', the AMI will select and use the Publish and Subscribe policy options that are applicable to MQSI V2 when sending publish, subscribe and unsubscribe requests. In addition, the AMI will insert 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)
```

If you wish to perform content-based publish/subscribe operations using MQSI 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 MQSeries Integrator Version 2. This allows the broker to recognise the broker being used. The AMI performs the required mapping of MQSeries Integrator Version 2 response values as necessary.

Migrating to MQSeries Integrator V2 from V1 and MQSeries Publish/Subscribe

MQSeries Integrator V2 will support applications written to use MQSI 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 MQSI V2 should have their Service Point definitions changed to a Service Type of 'MQSeries 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 MQSI V2.

Creating default MQSeries objects

The Application Messaging Interface makes use of default MQSeries 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 AS/400, start the the local queue manager by typing the following on the command line, where QMName is the name of your MQSeries queue manager: STRMOM MQMNAME(QMName)

Then run the default MQSC script by typing the following command: STRMQMMQSC SRCMBR(AMTSDFTS) SRCFILE(QMQMAMI/AMTMQSC) MQMNAME(QMName)

For OS/390, start the local queue manager, then use the CSQUTIL program to run the default MQSC script:

where hlq is the high level qualifier of your MQSeries installation, and QMGR is your queue manager name.

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 MQSeries 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)</pre>
```

where {QMName} is the name of your MQSeries queue manager and {Location} is the location of the amtsdfts.tst file.

460

The sample programs

1

Sample programs are provided to illustrate the use of the Application Messaging Interface.

It is recommended that you run one or more of the sample programs to verify that you have installed the Application Messaging Interface correctly.

If you are using the OS/390 platform, go to "Sample programs for OS/390" on page 464.

Sample programs for AS/400, UNIX, and Windows

There are ten basic sample programs for AS/400, 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 AS/400, 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	SendAndForget (for AS/400, SendForget)	SendAndForget
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	ReceiveFile (for AS/400, RcvFile)	ReceiveFile

To find the source code and the executables for the samples, see "Directory structure" on page 430 (AIX), page 433 (AS/400), page 438 (HP-UX), page 445 (Solaris), and page 448 (Windows).

The sample programs

Running the AS/400, UNIX, and Windows sample programs

Before you can run the sample programs on AS/400, UNIX, or Windows platforms, you must make a number of preparations. These are described in the following sections.

MQSeries objects

The sample programs require some MQSeries objects to be defined. To do this, use an MQSeries MQSC file, amtsamp.tst, which is shipped with the samples.

For AS/400, start the the local queue manager by typing the following on the command line:

STRMQM MQMNAME (QMName)

where QMName is the name of your MQSeries 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 MQSeries 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)</pre>
```

where {QMName} is the name of your MQSeries 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 (AS/400, UNIX, and Windows)" on page 450).

Modify the host file so that your MQSeries queue manager name, {QMName}, is known as defaultConnection.

Running the publish/subscribe samples

To run the AMI publish/subscribe samples, you need access to an MQSeries broker. This can be either MQSeries Publish/Subscribe or MQSeries 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 an MQSeries broker, publish/subscribe requests can only be issued remotely. In this situation, you must set up the appropriate MQSeries channels, and ensure that the remote queue manager and channels are started.

MQSeries Publish/Subscribe broker: To run the publish/subscribe samples with MQSeries 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 MQSeries queue manager.

. .

MQSeries Integrator Version 2: To run the publish/subscribe samples with MQSeries Integrator Version 2 you need to do the following:

 Start the broker and the Configuration Manager. Type the following at a command line, where {BrokerName} is is the name of your MQSeries Integrator Version 2 broker:

```
mqsistart {BrokerName}
mqsistart ConfigMgr
```

| |

I

I

I

1

1

- 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
```

For AS/400 only:

- a. In the properties for the MQInput node, select the "Advanced" properties pane and check the Convert check box.
- b. Set the Convert Encoding and Convert Coded Character Set ID to the native values used by MQSeries 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 MQSeries 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 429 (AIX), page (AS/400), page 437 (HP-UX), page 444 (Solaris), and page 447 (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.mg.amt.samples.SendAndForget
```

Running the sample programs (AS/400)

Executable sample programs are provided in the QMQMAMI library. To run C and C++ samples on the AS/400, 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
```

The sample programs

Sample programs for OS/390

There are ten basic sample programs in C for the OS/390 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 OS/390 ('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
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

To find the source code for the samples, see "Directory structure (OS/390)" on page 441.

Running the sample programs (OS/390)

Before you can run the sample programs on the OS/390 platform, there are a number of actions to take.

Building the sample programs

The samples for OS/390 are provided as source code only, so you must build them before you can run them. See "Building C applications" on page 29 and "COBOL applications on OS/390" on page 241.

MQSeries objects

The sample programs require some MQSeries objects to be defined. This can be done with an MQSeries MQSC file, AMTSAMP, which is shipped with the samples.

First start the local queue manager, as described in the MQSeries for OS/390 System Administration Guide. If you are using the CICS environment, ensure that the MQSeries 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 MQSeries utility program CSQUTIL. Following is a JCL fragment to help you run the utility:

```
//COMMAND EXEC PGM=CSQUTIL,PARM='QMGR'
//STEPLIB DD DSN=h1q.SCSQAUTH,DISP=SHR
// DD DSN=h1q.SQSCANLE,DISP=SHR
//AMTSAMP DD DSN=h1q.SCSQPROC(AMTSAMP),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COMMAND DDNAME(AMTSAMP)
/*
```

where hlq is the high level qualifier of your MQSeries installation, and 0MGR 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 (OS/390)" on page 453.

For batch, RRS-batch, and IMS programs (not CICS), copy the sample host file AMTHOST (UTF-8) or AMTHOST (EBCDIC) from hlq.SCSQPROC to another location, and modify it so that your MQSeries 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 (OS/390)" on page 453.

Running the publish/subscribe samples

To use the publish/subscribe samples, you need access to an MQSeries Publish/Subscribe broker. Because this is not available on OS/390, you must have an MQSeries queue manager and publish/subscribe broker running on another platform. In this situation, you must set up the appropriate MQSeries channels, and ensure that the remote queue manager and channels are started.

MQSeries Publish/Subscribe broker: To run the publish/subscribe samples with MQSeries Publish/Subscribe broker, you must start the broker. Type the following at a command line:

```
strmqbrk -m {QMName}
```

I

465

The sample programs

where {QMName} is the name of your MQSeries queue manager.

MQSeries Integrator Version 2: To run the publish/subscribe samples with MQSeries 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 is the name of your MQSeries Integrator Version 2 broker:

```
mgsistart {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 MQSeries 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

For further details, refer to the MQSeries 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 440.

File name input for the file transfer samples

There are 3 ways in MVS to specify the file name for the file transfer samples:

1. Use single quotes.

```
// PARM='MYTEST.FILE'
```

It will then obey MVS 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.

The 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.

Chapter 18. Defining services and policies

Definitions of services and policies 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 and policies, and to specify their attributes (see "The administration tool" on page 457).

This chapter contains:

- · "Services and policies"
- "Service definitions" on page 472
- "Policy definitions" on page 475

Services and policies

A repository file contains definitions for *policies* 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, 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 or policy 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 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 470).

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 and policies 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 475, 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 472, 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

Services and policies

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" on page 509), 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" on page 509). 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'.
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.
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 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 'MQSeries Integrator V2' to use the appropriate publish and subscribe policy options when sending publish, subscribe and unsubscribe requests to the MQSeries 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 MQSeries 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 an MQSeries 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 'MQSeries 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

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 'MQSeries 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
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 'MQSeries 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
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 'MQSeries 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 used by sending applications to prepare a message in the correct CCSID for the destination. 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 used by sending applications to prepare a message in the correct encoding for the destination. 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 MQSeries queue manager that does not provide native support for groups. (Currently, this only applies to MQSeries for OS/390 Version 2.x.)

Notes:

Ī

- 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 'MQSeries 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.

Service definitions

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

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.
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.
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.
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 '%'.

Policy definitions

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).
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	No	As Transport	The persistence set in the message, where Yes is persistent and No is not persistent.
	As Transport		When set to As Transport, the value from the underlying queue definition is used.
Expiry Interval	0-99999999 Unlimited	Unlimited	A period of time (in tenths of a second) after which the message will not be delivered.
Retry Count	0-99999999	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	0-99999999	1000	The interval (in milliseconds) between each retry.
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.
Report Type Exception	Selected Not selected	Not selected	When selected, Exception reports are required.

Table 15. Send attributes (continued)

Attribute	Values	Default	Comments
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. On Windows, HP-UX, AIX, and Sun Solaris, this is the end of a line. On OS/390, this is a record boundary. When set to Physical, the file is split into separate
			messages at boundaries that are determined by AMI.
Bind On Open	Yes No As Transport	As Transport	Bind On Open controls the binding of a service point to a particular instance of an MQSeries 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 an MQSeries 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 '%'.

Policy definitions

Receive attributes

Table 16. Receive attributes

	Attribute	Values	Default	Comments
I	Implicit Open	Selected Not selected	Selected	When selected, the queue is opened implicitly (must be selected for the C and COBOL high-level interfaces).
I	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	0-99999999 Unlimited	Unlimited	A period of time (in milliseconds) that the receive waits for a message to be available.
I	Wait Interval Read Only	Selected Not selected	Selected	When selected, an application cannot override the Wait Interval value in the policy object.
I	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 MQSeries transport queue object. If poison message handling is enabled during a receive request the AMI will handle it as follows:

If a poison message is successfully requeued to the backout-requeue queue (specified by the underlying MQSeries 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.

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 MQSeries 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 MQSeries Integrator Version 2.

479

Policy definitions

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 MQSeries 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.

Chapter 19. 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 (AS/400, UNIX, and Windows)"
- "Using trace (OS/390)" on page 489
- "When your AMI program fails" on page 492

Using trace (AS/400, 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 AS/400, 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 AS/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:
 - Select Main -> Control Panel on Windows NT and Windows 98
 - Edit your .profile file on UNIX systems

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.

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 MQSeries environment.

I

Using trace (AS/400, UNIX, and Windows)

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 to the trace environment variable used within the MQSeries 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 AS/400

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:

- setTraceLocation(location);
- 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 d:\output\bin\amITSR.exe <<< AMT trace >>> started at Sat Jun 12 08:28:33 1999

```
@(!) <<< *** Code Level is 1.0.0 *** >>>
   !(03787) BuildDate Jun 11 1999
   !(03787) Trace Level is 2
(03787)@08:28:33.728
   -->xmq xxxInitialize
   --->ObtainSystemCP
   !(03787) Code page is 437
  <----ObtainSystemCP (rc = 0)
  <--xmq xxxInitialize (rc = 0)
   -->amSessCreateX
   ---->amCheckAllBlanks()
   <---amCheckAllBlanks() (rc = 0)
```

```
---->amCheckValidName()
   <---amCheckValidName() (rc = 1)
   !(03787) Session name is: plenty
  ---->amHashTableCreate()
  <----amHashTableCreate() (rc = AM_ERR_OK)
   ---->amSessClearErrorCodes
  <---amSessClearErrorCodes (rc = 0)
  --->amMaSrvCreate
  !(03787) Service object created [9282320]
  <---amMaSrvCreate (rc = AM_ERR_OK)
   ---->amMaSrvSetSessionHandle
  !(03787) Object handle[9282320]
  <----amMaSrvSetSessionHandle (rc = AM ERR OK)
  ---->amHashTableAddHandle()
  <---amHashTableAddHandle() (rc = AM ERR OK)
   --->amMaSrvCreate
  !(03787) Service object created [9285144]
  <---amMaSrvCreate (rc = AM ERR OK)
   ---->amMaSrvSetSessionHandle
   !(03787) Object handle[9285144]
  <----amMaSrvSetSessionHandle (rc = AM_ERR OK)
  ---->amHashTableAddHandle()
  <---amHashTableAddHandle() (rc = AM_ERR_OK)</pre>
(03787)@08:28:33.738
   --->amMaSrvCreate
  !(03787) Service object created [9287968]
  <---amMaSrvCreate (rc = AM_ERR_OK)
   ---->amMaSrvSetSessionHandle
  !(03787) Object handle[9287968]
  <----amMaSrvSetSessionHandle (rc = AM ERR OK)
   --->amHashTableAddHandle()
  <---amHashTableAddHandle() (rc = AM_ERR_OK)
   --->amMaSrvCreate
  !(03787) Service object created [9290792]
  <---amMaSrvCreate (rc = AM_ERR_OK)
   ---->amMaSrvSetSessionHandle
   !(03787) Object handle[9290792]
   <----amMaSrvSetSessionHandle (rc = AM ERR OK)
```

```
---->amHashTableAddHandle()
<---amHashTableAddHandle() (rc = AM_ERR_OK)
---->amMaSrvCreate
!(03787) Service object created [9293616]
<---amMaSrvCreate (rc = AM_ERR_OK)
---->amMaSrvSetSessionHandle
!(03787) Object handle[9293616]
<---amMaSrvSetSessionHandle (rc = AM ERR OK)
---->amHashTableAddHandle()
<---amHashTableAddHandle() (rc = AM ERR OK)</pre>
--->amMaSrvCreate
!(03787) Service object created [9296440]
<---amMaSrvCreate (rc = AM ERR OK)
---->amMaSrvSetSessionHandle
!(03787) Object handle[9296440]
<----amMaSrvSetSessionHandle (rc = AM ERR OK)
---->amMaSrvSetSubReceiverHandle
!(03787) Object handle[9293616]
<----amMaSrvSetSubReceiverHandle (rc = AM ERR OK)
--->amMaMsgCreate
!(03787) message object created -[10420288]
<----amMaMsgCreate (rc = AM_ERR_OK)
---->amHashTableAddHandle()
<----amHashTableAddHandle() (rc = AM_ERR_OK)
--->amMaMsgCreate
!(03787) message object created -[10432440]
<---amMaMsgCreate (rc = AM ERR OK)
---->amHashTableAddHandle()
<----amHashTableAddHandle() (rc = AM_ERR_OK)
--->amMaPolCreate
!(03787) policy object created.
!(03787) policy object initialized.
<---amMaPolCreate (rc = AM ERR OK)
---->amHashTableAddHandle()
<----amHashTableAddHandle() (rc = AM_ERR_OK)
--->amMaPolCreate
!(03787) policy object created.
!(03787) policy object initialized.
<---amMaPolCreate (rc = AM ERR OK)
```

```
---->amHashTableAddHandle()
  <----amHashTableAddHandle() (rc = AM_ERR_OK)
  ---->amMaPolSetIntProps
  !(03787) Object handle[10446656]
  !(03787) [AMPOL IPR APR CON CNT] set to [0x1]
(03787)@08:28:33.748
   <----amMaPolSetIntProps (rc = AM ERR OK)
  ---->amMaPolSetStringProp
  !(03787) Object handle[10446656]
  !(03787) [AMPOL SPR APR MGR NAME] set to [plenty]
  <---amMaPolSetStringProp (rc = AM ERR OK)</pre>
   ---->amMaPolSetStringProp
  !(03787) Object handle[10446656]
   !(03787) [AMPOL_SPR_APR_CON_NAME] set to [plenty]
  <----amMaPolSetStringProp (rc = AM ERR OK)
  ---->amMaSrvSetStringProp
   !(03787) Object handle[9282320]
   !(03787) [AMSRV_SPR_QUEUE_NAME] set to [SYSTEM.DEFAULT.SENDER]
  <----amMaSrvSetStringProp (rc = AM ERR OK)
   ---->amMaSrvSetStringProp
  !(03787) Object handle[9285144]
  !(03787) [AMSRV SPR QUEUE NAME] set to []
  <---amMaSrvSetStringProp (rc = AM ERR OK)
   ---->amMaSrvSetStringProp
  !(03787) Object handle[9287968]
   !(03787) [AMSRV SPR QUEUE NAME] set to [SYSTEM.DEFAULT.RECEIVER]
  <---amMaSrvSetStringProp (rc = AM ERR OK)</pre>
  ---->amMaSrvSetStringProp
   !(03787) Object handle[9290792]
   !(03787) [AMSRV SPR QUEUE NAME] set to [SYSTEM.DEFAULT.PUBLISHER]
  <----amMaSrvSetStringProp (rc = AM ERR OK)
   ---->amMaSrvSetStringProp
   !(03787) Object handle[9293616]
   !(03787) [AMSRV SPR QUEUE NAME] set to [SYSTEM.DEFAULT.SUBSCRIBER]
  <----amMaSrvSetStringProp (rc = AM_ERR_OK)
   ---->amMaPolSetIntProps
   !(03787) Object handle[10451304]
   !(03787) [AMPOL_IPR_SMO_SYNCPOINT] set to [0xc030003]
  <----amMaPolSetIntProps (rc = AM_ERR_OK)
   ---->amMaPolSetIntProps
   !(03787) Object handle[10451304]
   !(03787) [AMPOL_IPR_RMO_SYNCPOINT] set to [0xd060002]
  <---amMaPolSetIntProps (rc = AM ERR OK)
   ---->amActivateFiles
   !(03787) No DATAPATH specified from API
```

```
!(03787) No repository FILE specified from API !(03787) Repository[H:\MQSeries\amt\\amt.xml]
!(03787) Repository ACTIVE
!(03787) No local host FILE specified from API
!(03787) Local Host[H:\MQSeries\amt\\amthost.xml]
!(03787) Local Host File ACTIVE
<---amActivateFiles (rc = 1)
---->amErrTranslate
<---amErrTranslate (rc = 0)
<--amSessCreateX (rc = 0)
```

Using trace (OS/390)

The AMI provides two types of trace on OS/390:

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:

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.

Using trace (OS/390)

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: 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 MQSeries for OS/390 Application GTF trace, allowing for AMI and MQSeries trace entries to be selected together in IPCS and formatted in a single, chronological, stream. Unlike MQSeries, 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
                                           ∥ .6..A8SNELLSamSe
+0000 00F63080 C1F8E2D5 C5D3D3E2 8194E285
                                           sClearErrorCodes
+0010 A2C39385 8199C599 999699C3 968485A2
+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
. . . . . . . . . . . . . . . . . . .
+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 OS/390 operator's console using an installation defined procedure. The chapter

Using trace (OS/390)

"Using trace for problem determination" in the MQSeries for OS/390 Problem Determination Guide describes a typical GTF start-up prompt/reply sequence. If AMI and MQSeries GTF trace entries are to be captured to the same dataset, the job names for both the AMI application and the MQSeries 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" on page 497.

In addition, if MQSeries is the reason for the failure, AMI supplies a secondary reason code. The secondary reason codes can be found in the MQSeries Application Programming Reference book.

First failure symptom report (AS/400, 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 (OS/390)

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
```

When your AMI program fails

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 MQSeries as a transport mechanism and so MQSeries error logs and trace information can provide useful information. See the MQSeries System Administration manual for details of how to activate these problem determination

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 MQSeries 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 MQSeries transport. The secondary reason code returned by the appropriate 'get last error' function for the object concerned will provide the related the MQSeries reason code. This error occurs most frequently during an attempt to open an underlying MQSeries 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 8. Appendixes

Appendix A. Reason 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" on page 509.

In some circumstances the AMI returns a secondary reason code that comes from MQSeries, the underlying transport layer. Please refer to the MQSeries Application Programming Reference manual for details of these reason codes.

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 "Unicode character conversion" on page 440 for more details, and see the OS/390 C/C++ Programming Guide for a list of the coded character sets supported under OS/390.

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.

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.

Reason code (warning)

1

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 occours 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 OS/390 fixed-length dataset to a UNIX file or between OS/390 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_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_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 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 (MQSeries) 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 OS/390 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 an invalid combination of options.

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 OS/390 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_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.

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 an invalid value.

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.

Reason code (failed)

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 OS/390 VSAM datasets, and OS/390 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 OS/390 datasets. On other platforms and for HFS files on OS/390, 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 OS/390 tried to perform a file transfer operation, which is invalid 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 invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC INVALID IF SERVICE OPEN

The receiver queue name could not be set because the receiver or subscriber service was open.

AMRC_INVALID_MSG_NAME

The specified message name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_POLICY_NAME

The specified policy name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_PUBLISHER_NAME

The specified publisher service name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_Q_NAME

The specified queue name was too long, or contained invalid characters.

AMRC_INVALID_RECEIVER_NAME

The specified receiver service name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_SENDER_NAME

The specified sender service name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_SESSION_NAME

The specified session name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC INVALID SUBSCRIBER NAME

The specified subscriber service name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_TRACE_LEVEL

A specified trace level was not valid.

Reason code (failed)

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 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.

AMRC NAME LEN PTR ERR

The specified name length pointer was not valid.

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_HANDLE_ERR

The specified policy handle was not valid.

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 O 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.

Reason code (failed)

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_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.

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 "Writing IMS applications using AMI" on page 421 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 (MQSeries) message transport layer. The message transport reason code can be obtained by the secondary reason

Reason code (failed)

code value returned from a 'GetLastError' request for the AMI object concerned. For more information, see "Common causes of problems" on page 493.

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.

Appendix B. Constants

This appendix lists the values of the named constants used by the functions described in this manual. For information about MQSeries constants not in this list, see the MQSeries Application Programming Reference manual and the MQSeries Programmable System Management manual.

The constants

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
                                  11
AMBRW LOCK
                                  2L
AMBRW FIRST
                                  4L
                                  8L
AMBRW NEXT
AMBRW_CURRENT
                                 16L
AMBRW RECEIVE CURRENT
                                 321
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 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 (OS/390
```

AMFB (Feedback codes)

AMFB_NONE	0L
AMFB EXPIRATION	1L
AMFB_COA	2L
AMFB COD	3L
AMFB ERROR	-1L

AMFMT (Format constants)

```
AMFMT NONE
AMFMT_RF_HEADER
                               "MQHRF
                              "MQSTR
AMFMT_STRING
                               "MQHRF2 "
AMFMT RF2 HEADER
```

AMGF and AMGRP (Group status constants)

```
AMGF_IN_GROUP
AMGF_FIRST
                      1L
                      2L
AMGF_LAST
                      4L
AMGRP MSG NOT IN GROUP
AMGRP_FIRST_MSG_IN_GROUP ( AMGF_IN_GROUP | AMGF_FIRST )
```

AMH (Handle constants)

```
AMH NULL HANDLE
                      (AMHANDLE) OL
AMH INVALID HANDLE
                      (AMHANDLE)-1L
```

AMLEN (String length constants)

```
AMLEN NULL TERM
                               -1L
                             256L
AMLEN_MAX_NAME_LENGTH
```

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 REQUEST	1L
AMMT REPLY	2L
AMMT REPORT	4L
AMMT DATAGRAM	8L

AMPS (Publish/subscribe)

Publish/Subscribe constants

Publish/subscribe tag names

AMPS COMMAND "MQPSCommand" AMPS COMP CODE "MQPSCompCode" AMPS_DELETE_OPTIONS "MQPSDelOpts" "MQPSErrorId" AMPS ERROR ID AMPS ERROR POS "MQPSErrorPos" AMPS PARAMETER_ID "MQPSParmId" AMPS PUBLICATION OPTIONS "MQPSPubOpts" AMPS TIMESTAMP "MQPSPubTime" "MQPSQMgrName" AMPS Q MGR NAME AMPS_Q_NAME "MQPSQName" AMPS_REASON "MQPSReason" "MQPSReasonText" AMPS REASON TEXT AMPS_REGISTRATION_OPTIONS AMPS_SEQUENCE_NUMBER "MQPSRegOpts" "MQPSSeqNum" AMPS STREAM NAME "MQPSStreamName" AMPS STRING DATA "MQPSStringData" AMPS TOPIC "MQPSTopic" "MQPSUserId" AMPS USER ID "MQPSFilter" AMPS FILTER AMPS_SUBSCRIPTION_POINT "MQPSSubPoint" "MQPSSequence" AMPS_SEQUENCE AMPS CONTROL "MQPSControl"

Publish/subscribe tag values

AMPS ANONYMOUS "Anon" AMPS CORREL ID AS ID "CorrelAsId" "DeregAll" AMPS DEREGISTER ALL 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 "PubOnRegOnly" AMPS DELETE PUBLICATION "DeletePub" AMPS DEREGISTER PUBLISHER "DeregPub" "DeregSub" AMPS DEREGISTER SUBSCRIBER AMPS_PUBLISH "Publish" AMPS_REGISTER_PUBLISHER "RegPub" "RegSub" AMPS REGISTER SUBSCRIBER 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" "Pers" AMPS PERSISTENT AMPS_NON_PERSISTENT "NonPers" AMPS_PERSISTENT_AS_PUBLISHER "PersAsPub" "PersAsQueue" AMPS PERSISTENT AS QUEUE AMPS CC OK "O" 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)

neason codes)	
AMRC_NONE AMRC_UNEXPECTED_ERR AMRC_INVALID_Q_NAME AMRC_INVALID_SENDER_NAME AMRC_INVALID_RECEIVER_NAME AMRC_INVALID_PUBLISHER_NAME AMRC_INVALID_SUBSCRIBER_NAME AMRC_INVALID_POLICY_NAME AMRC_INVALID_MSG_NAME AMRC_INVALID_SESSION_NAME	0 1 2 3 4 5 6 7 8 9
AMRC_INVALID_DIST_LIST_NAME AMRC_POLICY_HANDLE_ERR AMRC_SERVICE_HANDLE_ERR AMRC_MSG_HANDLE_ERR AMRC_SESSION_HANDLE_ERR AMRC_BROWSE_OPTIONS_ERR AMRC_INSUFFICIENT_MEMORY AMRC_WAIT_TIME_READ_ONLY AMRC_MSG_NOT_FOUND	10 11 12 13 14 15 16 17 18 19
AMRC_POLICY_NOT_FOUND AMRC_SENDER_NOT_UNIQUE AMRC_RECEIVER_NOT_UNIQUE AMRC_PUBLISHER_NOT_UNIQUE AMRC_SUBSCRIBER_NOT_UNIQUE AMRC_MSG_NOT_UNIQUE AMRC_POLICY_NOT_UNIQUE AMRC_DIST_LIST_NOT_UNIQUE AMRC_RECEIVE_BUFF_PTR_ERR AMRC_RECEIVE_BUFF_LEN_ERR	20 21 22 23 24 25 26 27 28 29
AMRC_SEND_DATA_PTR_ERR AMRC_SEND_DATA_LEN_ERR AMRC_INVALID_IF_SERVICE_OPEN AMRC_SERVICE_ALREADY_OPEN AMRC_DATA_SOURCE_NOT_UNIQUE AMRC_NO_MSG_AVAILABLE AMRC_SESSION_ALREADY_OPEN AMRC_SESSION_ALREADY_CLOSED AMRC_ELEM_NOT_FOUND AMRC_ELEM_COUNT_PTR_ERR	30 31 32 33 34 35 36 37 38 39
AMRC_ELEM_NAME_PTR_ERR AMRC_ELEM_NAME_LEN_ERR AMRC_ELEM_INDEX_ERR AMRC_ELEM_PTR_ERR AMRC_ELEM_STRUC_ERR AMRC_ELEM_STRUC_NAME_ERR AMRC_ELEM_STRUC_VALUE_ERR AMRC_ELEM_STRUC_VALUE_ERR AMRC_ELEM_STRUC_VALUE_BUFF_ERR AMRC_ELEM_STRUC_VALUE_BUFF_ERR AMRC_ELEM_STRUC_VALUE_BUFF_ERR AMRC_TRANSPORT_ERR	40 41 42 43 44 45 46 47 48 49
AMRC_TRANSPORT_WARNING AMRC_ENCODING_INCOMPLETE AMRC_ENCODING_MIXED AMRC_ENCODING_ERR AMRC_BEGIN_INVALID AMRC_NO_REPLY_TO_INFO AMRC_SERVICE_ALREADY_CLOSED	50 51 52 53 54 55 56

AMRC_SESSION_NOT_OPEN AMRC_DIST_LIST_INDEX_ERR AMRC_WAIT_TIME_ERR	57 58
AMRC_WAIT_TIME_ERR AMRC SERVICE NOT OPEN	59 60
AMRC HEADER TRUNCATED	61 62
AMRC_HEADER_INVALID AMRC_DATA_LEN_ERR	63
AMRC_BACKOUT_REQUEUE_ERR	64 65
AMRC_BACKOUT_LIMIT_ERR AMRC_COMMAND_ALREADY_EXISTS	66
AMRC_UNEXPECTED_RECEIVE_ERR	67 68
AMRC_UNEXPECTED_SEND_ERR	00
AMRC_SENDER_USAGE_ERR AMRC MSG TRUNCATED	70 71
AMRC_CLOSE_SESSION_ERR	72
AMRC_READ_OFFSET_ERR AMRC_RFH_ALREADY_EXISTS	73
AMRC_RFH_ALREADY_EXISTS AMRC_GROUP_STATUS_ERR	74 75
AMRC MSG ID LEN ERR	76
AMRC_MSG_ID_LEN_ERR AMRC_MSG_ID_PTR_ERR	77
AMRC_MSG_ID_BUFF_LEN_ERR AMRC_MSG_ID_BUFF_PTR_ERR	78 79
	73
AMRC_MSG_ID_LEN_PTR_ERR AMRC_CORREL_ID_LEN_ERR	80
AMRC_CORREL_ID_LEN_ERR AMRC_CORREL_ID_PTR_ERR	81 82
AMRC_CORREL_ID_BUFF_LEN_ERR AMRC_CORREL_ID_BUFF_PTR_ERR	83
AMRC_CORREL_ID_BUFF_PTR_ERR AMRC_CORREL_ID_LEN_PTR_ERR	84 85
AMRC_CORREL_ID_LEN_PTR_ERR AMRC FORMAT LEN ERR	86
AMRC_FORMAT_LEN_ERR AMRC_FORMAT_PTR_ERR	87
AMRC_FORMAT_BUFF_PTR_ERR AMRC_FORMAT_LEN_PTR_ERR	88 89
	03
AMRC_FORMAT_BUFF_LEN_ERR AMRC_NAME_BUFF_PTR_ERR	90
AMRC NAME LEN PTR ERR	91 92
AMRC_NAME_BUFF_LEN_ERR AMRC_Q_NAME_LEN_ERR	93
AMRC_Q_NAME_LEN_ERR AMRC_Q_NAME_PTR_ERR	94 95
AMRC_Q_NAME_BUFF_PTR_ERR	96
AMRC Q NAME BUFF PTR ERR AMRC Q NAME LEN PTR ERR	97
AMRC_Q_NAME_BUFF_LEN_ERR AMRC_WAIT_TIME_PTR_ERR	98 99
AMRC CCSID PTR ERR	100
AMRC ENCODING PTR ERR	101
AMRC DEFN TYPE PTR ERR	102
AMRC_CCSID_ERR AMRC_DATA_LEN_PTR_ERR	103 104
AMRC_GROUP_STATUS_PTR_ERR	105
AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR	106 107
AMRC_RESP_RECEIVER_HANDLE_ERR	108
AMRC_NOT_AUTHORIZED	109
AMRC_TRANSPORT_NOT_AVAILABLE	110
AMRC_BACKED_OUT	111 112
AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED	113
AMRC SERVICE FULL	114
AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED	115 116

AMRC_GROUP_BACKOUT_LIMIT_ERR AMRC_SENDER_COUNT_PTR_ERR AMRC_MULTIPLE_REASONS	117 118 119
AMRC_DATA_PTR_ERR AMRC_DATA_BUFF_LEN_ERR AMRC_DATA_BUFF_PTR_ERR	120 121 122 123 124
AMRC_COMMIT_INVALID AMRC_DATA_OFFSET_ERR AMRC_FILE_SYSTEM_ERR AMRC_FILE_ALREADY_EXISTS AMRC_REPORT_CODE_PTR_ERR AMRC_MSG_TYPE_PTR_ERR AMRC_FILE_FORMAT_CONVERTED AMRC_FILE_TRUNCATED AMRC_FILE_NOT_FOUND AMRC_NOT_A_FILE AMRC_FILE_NAME_LEN_ERR AMRC_FILE_NAME_PTR_ERR AMRC_CCSID_NOT_SUPPORTED AMRC_MSG_TYPE_NOT_REPORT AMRC_MSG_TYPE_NOT_REPORT AMRC_ELEM_STRUC_TYPE_BUFF_ERR AMRC_ELEM_STRUC_TYPE_BUFF_ERR	125 126 127 128 129 130 131 132 133 134 135 136 137 140 141 142 143 144
AMRC_FILE_NOT_WRITTEN AMRC_FILE_FORMAT_NOT_SUPPORTED	145 146
AMRC_CONN_NAME_NOT_FOUND AMRC_HOST_FILE_NOT_FOUND AMRC_HOST_FILE_NAME_ERR AMRC_HOST_FILE_ERR AMRC_POLICY_NOT_IN_REPOS AMRC_SENDER_NOT_IN_REPOS AMRC_RECEIVER_NOT_IN_REPOS AMRC_DIST_LIST_NOT_IN_REPOS AMRC_PUBLISHER_NOT_IN_REPOS	400 401 402 403 404 405 406 407 408 409 410
AMRC_REPOS_FILENAME_ERR AMRC_REPOS_WARNING AMRC_REPOS_ERR AMRC_REPOS_NOT_FOUND AMRC_TRANSPORT_LIBRARY_ERR AMRC_HOST_CACHE_ERR AMRC_REPOS_CACHE_ERR	411 414 415 416 418 419 420 421 422 423

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" "SYSTEM.DEFAULT.RECEIVER" AMSD RCV AMSD RCV MSG "SYSTEM.DEFAULT.RECEIVE.MESSAGE" AMSD RSP SND "SYSTEM.DEFAULT.RESPONSE.SENDER" AMSD SND "SYSTEM.DEFAULT.SENDER" "SYSTEM.DEFAULT.SEND.MESSAGE" AMSD SND MSG AMSD SESSION NAME "SYSTEM.DEFAULT.SESSION" "SYSTEM.DEFAULT.SUBSCRIBER" ${\sf AMSD_SUB}$ AMSD_SUB_SND "SYSTEM.DEFAULT.SUBSCRIBER" AMSD SUB RCV "SYSTEM.DEFAULT.SUBSCRIBER.RECEIVER" "SYSTEM.DEFAULT.SYNCPOINT.POLICY" AMSD_SYNC_POINT_POL

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

Appendix C. 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.

Notices

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.

Trademarks

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX AS/400 BookManager

CICS **IBM IBMLink** Language Environment **MQSeries** OS/400 RACF OS/390

VisualAge VSE/ESA

Java, JDK and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, Visual C++ and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, or service names may be the trademarks or service marks of others.

Glossary of terms and abbreviations

This glossary defines terms and abbreviations used in this book. If you do not find the term you are looking for, see the Index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

B

broker. See message broker.

C

connection. An AMI connection maps a logical queue manager name in a policy to a real queue manager name. This allows applications running on different nodes to use the same policy to connect to different queue managers.

correlation identifier. This is used as a key to a message, for example to correlate a response message with a request message. The AMI normally sets this in a response message by copying the message identifier from the request message. See also *request/response* and *selection message*.

D

datagram. The simplest message that MQSeries supports. Also known as *send-and-forget*. This type of message does not require a reply. Compare with *request/response*.

distribution list. An AMI service. It contains a list of sender services, enabling a message to be sent to multiple destinations in one operation.

Ε

Extensible Markup Language (XML). A W3C standard for the representation of data.

F

filter. An expression that is applied to the content of a message to determine how the message is to be processed. See also *subscription filter*.

Ĺ

local host file. Defines the mapping from a logical connection name to a real MQSeries queue manager on the local machine.

M

message. A message defines what is sent from one program to another in an AMI application. See also *service* and *policy*.

message broker. A set of execution processes hosting one or more message flows.

message descriptor (MQMD). Control information describing the message format and properties that is carried as part of an MQSeries message.

message identifier. An identifier for the message. It is usually unique, and typically it is generated by the message transport (MQSeries).

message object. An AMI object. It contains attributes of the message, such as the message identifier and correlation identifier, and options that are used when sending or receiving the message (most of which come from the policy definition). It can also contain the message data.

message queue. See queue.

message queue interface (MQI). The programming interface provided by MQSeries queue managers. It allows application programs to access message queuing services. The AMI provides a simpler interface to these services.

MQRFH header. Header added to an MQSeries message to carry control information, typically for use by a broker (for example, in a publish/subscribe system).

P

point-to-point. Style of messaging application in which the sending application knows the destination of the message. Compare with *publish/subscribe*.

policy. A policy defines how a message is sent in an AMI application. It encapsulates many of the options available in the MQI. Its definition can be stored in a repository. See also *service*.

publish/subscribe. Style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers) using a broker. Compare with *point-to-point*. See also *topic*.

publisher. (1) An AMI service. It contains a sender service where the destination is a publish/subscribe

Glossary

broker. (2) An application that makes information about a specified topic available to a broker in a publish/subscribe system.

Q

queue. An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages: they point to other queues, or can be used as models for dynamic queues.

queue manager. A system program that provides queuing services to applications. It provides an application programming interface (the MQI) so that programs can access messages on the queues that the queue manager owns.

R

receiver. An AMI service. It represents a source (such as an MQSeries queue) from which messages are received. Its definition is stored in a repository as a service point.

repository. A repository provides definitions for services and policies. If the name of a service or policy is not found in the repository, or an AMI application does not have a repository, the definitions built into the AMI are used. See also *repository file*.

repository file. File that stores repository definitions in XML (Extensible Markup Language) format.

request/response. Type of messaging application in which a request message is used to request a response from another application. Compare with *datagram*. See also *response sender* and *selection message*.

response sender. A special type of sender service that is used to send a response to a request message. It must use the definition built into the AMI, so it must not be defined in the repository.

S

selection message. A message object that is used to selectively receive a message by specifying its correlation identifier. Used in request/response messaging to correlate a response message with its request message.

send-and-forget. See datagram.

sender. An AMI service. It represents a destination (such as an MQSeries queue) to which messages are sent. Its definition is stored in a repository as a service point.

service. A service defines where a message is sent in an AMI application. Senders, receivers, distribution lists, publishers, and subscribers are all types of service. Their definitions can be stored in a repository. See also *policy*.

service point. The definition in a repository of a sender or receiver service.

session. An AMI object. It creates and manages all other AMI objects (message, service, policy and connection objects), and it provides the scope for a unit of work when transactional processing is used.

subscriber. (1) An AMI service. It 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. (2) An application that requests information about a specified topic from a publish/subscribe broker.

subscription filter. A predicate that specifies a subset of messages to be delivered to a particular subscriber.

Т

topic. A character string that describes the nature of the data that is being published in a publish/subscribe system.

W

W3C. World Wide Web Consortium. An international industry consortium set up to develop commmon protocols to promote evolution and interoperability of the World Wide Web.

X

XML. Extensible Markup Language.

Bibliography

This section describes the documentation available for all current MQSeries products.

MQSeries cross-platform publications

Most of these publications, which are sometimes referred to as the MQSeries "family" books, apply to all MQSeries Level 2 products. The latest MQSeries Level 2 products are:

- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
 - MQSeries for AT&T GIS UNIX, V2.2
 - MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1
 - MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
 - MQSeries for OS/2 Warp, V5.1
 - MQSeries for OS/390, V5.2
 - MQSeries for SINIX and DC/OSx, V2.2
- MQSeries for Sun Solaris, V5.2
 - MQSeries for Sun Solaris, Intel Platform Edition, V5.1
 - MQSeries for Tandem NonStop Kernel, V2.2.0.1
- MQSeries for VSE/ESA, V2.1.1
 - MQSeries for Windows, V2.0
 - MQSeries for Windows, V2.1
- MQSeries for Windows NT and Windows 2000,
 V5.2

The MQSeries cross-platform publications are:

- MQSeries Brochure, G511-1908
- An Introduction to Messaging and Queuing, GC33-0805
- MQSeries Intercommunication, SC33-1872
- MQSeries Queue Manager Clusters, SC34-5349
- MQSeries Clients, GC33-1632
- MQSeries System Administration, SC33-1873
- MQSeries MQSC Command Reference, SC33-1369
- MQSeries Event Monitoring, SC34-5760
- MQSeries Programmable System Management, SC33-1482
- MQSeries Administration Interface Programming Guide and Reference, SC34-5390
- MQSeries Messages, GC33-1876
- MQSeries Application Programming Guide, SC33-0807

- MQSeries Application Programming Reference, SC33-1673
- MQSeries Programming Interfaces Reference Summary, SX33-6095
- MQSeries Using C++, SC33-1877
- MQSeries Using Java, SC34-5456
- MQSeries Application Messaging Interface, SC34-5604

MQSeries platform-specific publications

Each MQSeries product is documented in at least one platform-specific publication, in addition to the MQSeries family books.

MQSeries for AIX, V5.2

MQSeries for AIX Quick Beginnings, GC33-1867

MQSeries for AS/400, V5.2

MQSeries for AS/400 Quick Beginnings, GC34-5557

MQSeries for AS/400 System Administration, SC34-5558

MQSeries for AS/400 Application Programming Reference (ILE RPG), SC34-5559

MQSeries for AT&T GIS UNIX, V2.2

MQSeries for AT&T GIS UNIX System Management Guide, SC33-1642

MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1

MQSeries for Compaq (DIGITAL) OpenVMS System Management Guide, GC33-1791

MQSeries for Compaq Tru64 UNIX, V5.1

MQSeries for Compaq Tru64 UNIX Quick Beginnings, GC34-5684

MQSeries for HP-UX, V5.2

MQSeries for HP-UX Quick Beginnings, GC33-1869

MQSeries for Linux, V5.2

MQSeries for Linux Quick Beginnings, GC34-5691

Bibliography

MQSeries for OS/2 Warp, V5.1

MOSeries for OS/2 Warp Ouick Beginnings, GC33-1868

MQSeries for OS/390, V5.2

MQSeries for OS/390 Concepts and Planning Guide, GC34-5650

MQSeries for OS/390 System Setup Guide, SC34-5651

MQSeries for OS/390 System Administration Guide, SC34-5652

MQSeries for OS/390 Problem Determination Guide, GC34-5892

MQSeries for OS/390 Messages and Codes, GC34-5891

MQSeries for OS/390 Licensed Program Specifications, GC34-5893

MQSeries for OS/390 Program Directory

MQSeries link for R/3, Version 1.2

MQSeries link for R/3 User's Guide, GC33-1934

MQSeries for SINIX and DC/OSx, V2.2

MQSeries for SINIX and DC/OSx System Management Guide, GC33-1768

MQSeries for Sun Solaris, V5.2

MQSeries for Sun Solaris Quick Beginnings, GC33-1870

MQSeries for Sun Solaris, Intel Platform Edition, V5.1

MQSeries for Sun Solaris, Intel Platform Edition Quick Beginnings, GC34-5851

MQSeries for Tandem NonStop Kernel, V2.2.0.1

MQSeries for Tandem NonStop Kernel System Management Guide, GC33-1893

MQSeries for VSE/ESA, V2.1.1

MQSeries for VSE/ESA[™] Licensed Program Specifications, GC34-5365 MQSeries for VSE/ESA System Management Guide, GC34-5364

MQSeries for Windows, V2.0

MQSeries for Windows User's Guide, GC33-1822

MQSeries for Windows, V2.1

MQSeries for Windows User's Guide, GC33-1965

MQSeries for Windows NT and Windows 2000, V5.2

MQSeries for Windows NT Quick Beginnings, GC34-5389 MQSeries for Windows NT Using the Component Object Model Interface, SC34-5387 MQSeries LotusScript Extension, SC34-5404

Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

HTML format

Relevant MQSeries documentation is provided in HTML format with these MQSeries products:

- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
 - MQSeries for OS/2 Warp, V5.1
 - MQSeries for OS/390, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Windows NT and Windows 2000, V5.2 (compiled HTML)
 - MQSeries link for R/3, V1.2

The MQSeries books are also available in HTML format from the MQSeries product family Web site at:

http://www.ibm.com/software/mqseries/

Portable Document Format (PDF)

PDF files can be viewed and printed using the Adobe Acrobat Reader.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. Web site

http://www.adobe.com/

PDF versions of relevant MQSeries books are supplied with these MQSeries products:

- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
 - MQSeries for Compaq Tru64 UNIX, V5.1
 - MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
 - MQSeries for OS/2 Warp, V5.1
 - MQSeries for OS/390, V5.2

- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform 1 Edition, V5.1
- MQSeries for Windows NT and Windows 2000, V5.2
 - MQSeries link for R/3, V1.2

PDF versions of all current MQSeries books are also available from the MQSeries product family Web site at:

http://www.ibm.com/software/mgseries/

BookManager® format

The MQSeries library is supplied in IBM BookManager format on a variety of online library collection kits, including the Transaction Processing and Data collection kit, SK2T-0730. You can view the softcopy books in IBM BookManager format using the following IBM licensed programs:

BookManager READ/2 BookManager READ/6000 BookManager READ/DOS BookManager READ/MVS BookManager READ/VM

BookManager READ for Windows

PostScript format

The MQSeries library is provided in PostScript (.PS) format with many MQSeries Version 2 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

Windows Help format

The MQSeries for Windows User's Guide is provided in Windows Help format with MQSeries for Windows, Version 2.0 and MQSeries for Windows, Version 2.1.

MQSeries information available on the Internet

The MQSeries product family Web site is at: http://www.ibm.com/software/mqseries/

By following links from this Web site you can:

- · Obtain latest information about the MQSeries product family.
- · Access the MQSeries books in HTML and PDF formats.
- Download an MQSeries SupportPac[™].

MQSeries on the Internet

Index

_	A Division Time (T. 1) (1.11)	. F	
A	AmDistributionList (Java) (continued)	± , , ,	
Accept Direct Requests attribute 480	sendFile 404 AMDLCL 332	getReasonCode 220 getSource 220	
Accept Truncated Message attribute 478	AMDLCLEC 332	overview 181	
addElement	AMDLGTLE 332	toString 220	
AmMessage (C++) 191	AMDLGTNA 333	using 157	
AmMessage (Java) 389	AMDLGTSC 333	AmException (Java)	
addFilter	AMDLGTSH 334	getClassName 415	
AmMessage (C++) 191	AMDLOP 334	getCompletionCode 415	
AmMessage (Java) 390	AMDLSN 335	getMethodName 415	
addTopic	AMDLSNFL 335	getReasonCode 415	
AmMessage (C++) 191	amDstClearErrorCodes 126	getSource 415	
AmMessage (Java) 390	amDstClose 126	overview 381	
administration tool 457 AIX	amDstGetLastError 127	toString 415	
C++ applications 161	amDstGetName 127	using 363	
C applications 30	amDstGetSenderCount 128	AMFB constants 510	
installation 428	amDstGetSenderHandle 128	AMFMT constants 510	
Java applications 366	amDstOpen 128	AMGRR constants 510	
prerequisites 425	amDstSend 129	AMIL constants 510	
AMB constants 509	amDstSendFile 130	AMH constants 510 AMHBACK 247	
amBackout 40	AMDT constants 509	AMHBEGIN 248	
amBegin 41	AMELEM structure 24, 237	AMHBRMS 249	
amBrowseMsg 42	AmElement (C++)	AMHCMIT 251	
AMBRW constants 509	constructor 215	AMHINIT 252	
AmBytes (C++)	getName 215	AMHPB 253	
cmp 213	getValue 215 getVersion 215	AMHRCFL 254	
constructors 213	overview 179	AMHRCMS 256	
cpy 214	setVersion 215	AMHRCPB 258	
dataPtr 214	toString 215	AMHRCRQ 260	
destructor 214	using 157	AMHSB 266	
length 214	AmElement (Java)	AMHSNFL 262	
operators 214 overview 179	constructor 412	AMHSNMS 263	
pad 214	getName 412	AMHSNRQ 264	
AMCC constants 509	getValue 412	AMHSNRS 265	
amCommit 44	getVersion 412	AMHTERM 267	
AmConstants (Java) 411	overview 380	AMHUN 268	
AMDEF constants 509	setVersion 412	amInitialize 45	
AmDistributionList (C++)	toString 412	AMLEN constants 510	
clearErrorCodes 205	using 362	AMMCD constants 510	
close 205	AMENC constants 510	AmMessage (C++)	
enableWarnings 205	AmErrorException (C++)	addElement 191	
getLastErrorStatus 205	getClassName 221	addFilter 191	
getName 205	getCompletionCode 221	addTopic 191	
getSender 205	getMethodName 221	clearErrorCodes 191	
getSenderCount 205	getReasonCode 221	deleteElement 191	
open 205	getSource 221 overview 181	deleteFilter 191	
overview 175 send 206	toString 221	deleteNamedElement 192	
send 200 sendFile 206	AmErrorException (Java)	deleteTopic 192	
AmDistributionList (Java)	getClassName 416	enableWarnings 192 getCCSID 192	
clearErrorCodes 403	getCompletionCode 416	getCorrelationId 192	
close 403	getMethodName 416	getDataLength 192	
enableWarnings 403	getReasonCode 416	getDataOffset 192	
getLastErrorStatus 403	getSource 416	getElement 193	
getName 403	overview 381	getElementCCSID 193	
getSender 403	toString 416	getElementCount 193	
getSenderCount 403	AmException (C++)	getEncoding 193	
open 403	getClassName 220	getFilter 193	
overview 376	getCompletionCode 220	getFilterCount 193	
send 404	getMethodName 220	getFormat 193	

AmMessage (C++) (continued)	AMMSDEFI 300	AMMSRS 311
getGroupStatus 194	AMMSDENE 300	AMMSSELC 313
getLastErrorStatus 194	AMMSDETO 301	AMMSSTCC 312
getMessageId 194	amMsgAddElement 91	AMMSSTCI 312
getName 194	amMsgAddFilter 91	AMMSSTDO 312
getNamedElement 194	AmMsgAddStreamName 108	AMMSSTEN 313
getNamedElementCount 194	amMsgAddTopic 92	AMMSSTFO 314
getReportCode 195	amMsgClearErrorCodes 92	AMMSSTGS 314
getTopic 195	amMsgDeleteElement 92	AMMSWRBY 315
getTopicCount 195	amMsgDeleteFilter 93	AMMT constants 510
getType 195	amMsgDeleteNamedElement 93	AmObject (C++)
overview 171	amMsgDeleteTopic 94	clearErrorCodes 216
readBytes 195	AMMSGELC 301	getLastErrorStatus 216
reset 195	amMsgGetCCSID 94	getName 216
setCCSID 196	amMsgGetCorrelId 94	overview 179
setCorrelationId 196	amMsgGetDataLength 95	AmObject (Java)
setDataOffset 196	amMsgGetDataOffset 95	clearErrorCodes 413
setElementCCSID 196	amMsgGetElement 95	getLastErrorStatus 413
setEncoding 196	amMsgGetElementCCSID 96	getName 413
setFormat 197	amMsgGetElementCount 96	overview 380
setGroupStatus 197	amMsgGetEncoding 96	AMPBCL 337
writeBytes 197	amMsgGetFilter 97	AMPBCLEC 337
AmMessage (Java)	amMsgGetFilterCount 97	AMPBGTCC 337
addElement 389	amMsgGetFormat 98	AMPBGTEN 338
addFilter 390	amMsgGetGroupStatus 98	AMPBGTLE 338
addTopic 390	amMsgGetLastError 99	AMPBGTNA 339
clearErrorCodes 390	amMsgGetMsgId 99	AMPBOP 339
deleteElement 390	amMsgGetName 100	AMPBPB 340
deleteFilter 390	amMsgGetNamedElement 100	AMPOCLEC 348
deleteNamedElement 390	amMsgGetNamedElementCount 101	AMPOGTLE 348
deleteTopic 391	AmMsgGetPubTimeStamp 108	AMPOGTNA 349
enableWarnings 391	amMsgGetReportCode 101	AMPOGTWT 349
getCCSID 391	AmMsgGetStreamName 109	amPolClearErrorCodes 142
getCorrelationId 391	amMsgGetTopic 102	amPolGetLastError 142
getDataLength 391	amMsgGetTopicCount 103	amPolGetName 143
getDataOffset 391	amMsgGetType 102	amPolGetWaitTime 143
getElement 391	amMsgReadBytes 103	AmPolicy (C++)
getElementCount 391	amMsgReset 103	clearErrorCodes 212
getEncoding 392	amMsgSetCCSID 104	enableWarnings 212
getFilter 392	amMsgSetCorrelId 104	getLastErrorStatus 212
getFilterCount 392	amMsgSetDataOffset 104	getName 212
getFormat 392	amMsgSetElementCCSID 105	getWaitTime 212
getGroupStatus 392	amMsgSetEncoding 105	overview 178
getLastErrorStatus 392	amMsgSetFormat 106	setWaitTime 212
getMessageId 393	amMsgSetGroupStatus 106 AMMSGTCC 301	AmPolicy (Java)
getName 393	AMMSGTCL 301	clearErrorCodes 410
getNamedElement 393	AMMSGTDL 302	enableWarnings 410
getNamedElementCount 393	AMMSGTDC 302	getLastErrorStatus 410
getReportCode 393	AMMSGTEC 303	getName 410 getWaitTime 410
getTopic 393	AMMSGTEL 303	overview 379
getTopicCount 393	AMMSGTEN 304	setWaitTime 410
getType 394	AMMSGTFC 304	amPolSetWaitTime 143
overview 372	AMMSGTFI 305	
readBytes 394 reset 394	AMMSGTFO 305	AMPOSTWT 350
	AMMSGTGS 306	AMPS constants 511
setCCSID 394 setCorrelationId 394		amPubClearErrorCodes 131
setDataOffset 394	AMMSGTLE 306	amPubClose 131
	AMMSGTMI 307	amPubGetCCSID 131
setEncoding 395 setFormat 395	AMMSGTNA 307	amPubGetEncoding 132
setGroupStatus 395	AMMSGTNC 308	amPubGetLastError 132
writeBytes 395	AMMSGTNE 308	amPubGetName 133
AMMSADEL 298	AMMSGTRC 309	amPublish 46
	AMMSGTTC 310	AmPublisher (C++)
AMMEADEL 200	A A A A A A A A A A A A A A A A A A A	, ,
AMMSADFI 298	AMMSGTTO 309	clearErrorCodes 207
AMMSADTO 299	AMMSGTTY 310	clearErrorCodes 207 close 207
AMMSADTO 299 AMMSCLEC 299	AMMSGTTY 310 amMsgWriteBytes 107	clearErrorCodes 207 close 207 enableWarnings 207
AMMSADTO 299	AMMSGTTY 310	clearErrorCodes 207 close 207

A D 11:1 (C) ((' 1)	A D . (T) (A C 1 (T) ((' 1)
AmPublisher (C++) (continued)	AmReceiver (Java) (continued)	AmSender (Java) (continued)
getEncoding 207	getDefinitionType 400	enableWarnings 396
getLastErrorStatus 207	getLastErrorStatus 401	getCCSID 396
getName 207	getName 401	getEncoding 396
open 208	getQueueName 401	getLastErrorStatus 397
overview 176	open 401	getName 397
publish 208	overview 375	open 397
1		±
AmPublisher (Java)	receive 401	overview 374
clearErrorCodes 405	setQueueName 402	send 397
close 405	amReceiveRequest 53	sendFile 398
enableWarnings 405	AMSBCL 341	amSendFile 55
getCCSID 405	AMSBCLEC 341	amSendMsg 56
getEncoding 405	AMSBGTCC 342	amSendRequest 57
getLastErrorStatus 405	AMSBGTDT 342	amSendResponse 58
getName 405	AMSBGTEN 343	AMSEOP 296
open 406	AMSBGTLE 343	AMSERB 296
overview 377	AMSBGTNA 344	
		amSesBegin 78
publish 406	AMSBGTQN 344	amSesClearErrorCodes 78
amPubOpen 133	AMSBOP 345	amSesClose 79
amPubPublish 134	AMSBRC 345	amSesCommit 79
AMRC constants 512	AMSBSB 346	amSesCreate 79
AMRCBR 322	AMSBSTQN 346	amSesCreateDistList 80
AMRCBRSE 324	AMSBUN 347	amSesCreateMessage 80
AMRCCL 326	AMSD constants 515	amSesCreatePolicy 80
AMRCCLEC 325	AMSEBG 284	amSesCreatePublisher 81
AMRCGTDT 326		amSesCreateReceiver 81
	AMSECL 285	
AMRCGTLE 327	AMSECLEC 284	amSesCreateSender 82
AMRCGTNA 327	AMSECM 285	amSesCreateSubscriber 82
AMRCGTQN 328	AMSECR 286	amSesDelete 83
AMRCOP 328	AMSECRDL 286	amSesDeleteDistList 83
AMRCRC 329	AMSECRMS 287	amSesDeleteMessage 83
AMRCRCFL 330	AMSECRPB 288	amSesDeletePolicy 84
AMRCSTQN 331	AMSECRPO 287	amSesDeletePublisher 84
amRcvBrowse 115	AMSECRRC 288	amSesDeleteReceiver 84
amRcvBrowseSelect 117	AMSECRSB 289	amSesDeleteSender 85
amRcvClearErrorCodes 118	AMSECRSN 289	
		amSesDeleteSubscriber 85
amRcvClose 119	AMSEDL 290	amSesGetDistListHandle 85
amRcvGetDefnType 119	AMSEDLDL 290	amSesGetLastError 86
amRcvGetLastError 120	AMSEDLMS 290	amSesGetMessageHandle 86
amRcvGetName 120	AMSEDLPB 291	amSesGetPolicyHandle 87
amRcvGetQueueName 121	AMSEDLPO 291	amSesGetPublisherHandle 87
amRcvOpen 121	AMSEDLRC 291	amSesGetReceiverHandle 87
amRcvReceive 122	AMSEDLSB 292	
amRcvReceiveFile 124	AMSEDLSN 292	amSesGetSenderHandle 88
	AMSEGHDL 292	amSesGetSubscriberHandle 88
amRcvSetQueueName 125		amSesOpen 88
amReceiveFile 47	AMSEGHMS 293	amSesRollback 89
amReceiveMsg 49	AMSEGHPB 294	AmSession (C++)
amReceivePublication 51	AMSEGHPO 294	begin 186
AmReceiver (C++)	AMSEGHRC 294	clearErrorCodes 186
browse 201	AMSEGHSB 295	close 186
clearErrorCodes 202	AMSEGHSN 295	
	AMSEGTLE 293	commit 186
close 202		createDistributionList 186
enableWarnings 202	AmSender (C++)	createMessage 187
getDefinitionType 202	clearErrorCodes 198	createPolicy 187
getLastErrorStatus 203	close 198	createPublisher 187
getName 203	enableWarnings 198	createReceiver 187
getQueueName 203	getCCSID 198	createSender 187
open 203	getEncoding 198	createSubscriber 187
overview 174	getLastErrorStatus 199	deleteDistributionList 188
receive 203	getName 199	deleteMessage 188
receiveFile 204, 402	open 199	deletePolicy 188
setQueueName 204	overview 173	deletePublisher 188
-		
AmReceiver (Java)	send 199	deleteReceiver 188
browse 399	sendFile 200	deleteSender 188
clearErrorCodes 400	AmSender (Java)	deleteSubscriber 188
close 400	clearErrorCodes 396	enableWarnings 188
enableWarnings 400	close 396	getLastErrorStatus 189

AmSession (C++) (continued)	amSndSend 113	AmSubscriber (Java)
		- · · · · · · · · · · · · · · · · · · ·
getName 189	amSndSendFile 114	clearErrorCodes 407
getTraceLevel 189	AMSNGTCC 317	close 407
getTraceLocation 189	AMSNGTEN 317	enableWarnings 407
open 189	AMSNGTLE 318	getCCSID 407
overview 169	AMSNGTNA 318	getDefinitionType 407
rollback 189	AMSNOP 319	getEncoding 407
transaction coordination 186	AMSNSN 319	getLastErrorStatus 407
unit of work 186	AMSNSNFL 320	0
		getName 408
AmSession (Java)	AmStatus (C++)	getQueueName 408
begin 386	constructor 217	open 408
clearErrorCodes 386	getCompletionCode 217	overview 378
close 386	getReasonCode 217	receive 408
commit 386	getReasonCode2 217	setQueueName 408
createDistributionList 386	overview 179	subscribe 409
		unsubscribe 409
createMessage 387	toString 217	
createPolicy 387	using 157	amSubSetQueueName 139
createPublisher 387	AmStatus (Java)	amSubSubscribe 140
createReceiver 387	constructor 414	amSubUnsubscribe 141
createSender 387	getCompletionCode 414	amtc.h header 29
createSubscriber 387	getReasonCode 414	AMTELEMV and AMTELEML
enableWarnings 388	getReasonCode2 414	copybooks 237
	overview 380	amTerminate 60
getLastErrorStatus 388		
getName 388	toString 414	amUnsubscribe 61
getTraceLevel 388	using 363	AmWarningException (C++)
getTraceLocation 388	AmString (C++)	getClassName 222
open 388	cat 218	getCompletionCode 222
overview 371	cmp 218	getMethodName 222
rollback 388	constructors 218	getReasonCode 222
		getSource 222
transaction coordination 386	contains 218	_
unit of work 386	cpy 218	overview 181
AmSessionFactory (C++)	destructor 218	toString 222
constructors 184	length 219	AmWarningException (Java)
createSession 184	operators 219	getClassName 417
deleteSession 184	overview 180	getCompletionCode 417
	pad 219	getMethodName 417
getFactoryName 184	1	9
getLocalHost 184	split 219	getReasonCode 417
getRepository 184	strip 219	getSource 417
getTraceLevel 184	text 219	overview 381
getTraceLocation 184	truncate 219	toString 417
overview 168	amSubClearErrorCodes 135	AMWT constants 515
setLocalHost 185	amSubClose 135	Anonymous Registration attribute 479
setRepository 185		480
1 ,	amSubGetCCSID 135	
setTraceLevel 185	amSubGetDefnType 136	appearance of text in this book xx
setTraceLocation 185	amSubGetEncoding 136	Application Group attribute 476
AmSessionFactory (Java)	amSubGetLastError 137	application messaging interface
constructor 384	amSubGetName 137	basic model 7
createSession 384	amSubGetQueueName 138	description 4
getFactoryName 384	-	interoperability 3
getLocalHost 384	amSubOpen 138	± *
ĕ	amSubReceive 139	introduction 3
getRepository 384	amSubscribe 59	main components 3
getTraceLevel 384	AmSubscriber (C++)	programming languages 4
getTraceLocation 384	clearErrorCodes 209	receiving messages 3
overview 370		sending messages 3
setLocalHost 384	close 209	applications, building
setRepository 385	enableWarnings 209	
setTraceLevel 385	getCCSID 209	C 29
	getDefinitionType 209	C++ 161
setTraceLocation 385	getEncoding 209	COBOL 241
AMSNCL 317	getLastErrorStatus 209	Java 366
AMSNCLEC 316	getName 210	applications, writing
amSndClearErrorCodes 110	getQueueName 210	C 16
amSndClose 110		
	open 210	C++ 151
amSndGetCCSID 111	overview 177	COBOL 230
amSndGetEncoding 111	receive 210	Java 357
amSndGetLastError 111	setQueueName 210	AS/400
amSndGetName 112	subscribe 211	C++ applications 162
amSndOpen 112	unsubscribe 211	C applications 31
	andapperine All	c applications of

AS/400 (continued)	cache, repository and local host	Connection Name attribute 475	
installation 432	(OS/390) 454	Connection Type attribute 475	
Java applications 366	CCSID attribute 472	constants	
prerequisites 425	class (C++)	Boolean 509	
sample programs 461	base 149, 183	browse 509	
attributes, policy	exception 150, 183	completion codes 509	
general 476	helper 150, 183	definition type 509	
initialization 475	class (Java)	encoding 510	
publish 480	base 355, 383	feedback codes 510	
receive 478	exception 356, 383	format 510	
send 476	helper 356, 383	group status 510	
subscribe 479	clearErrorCodes	handle 510	
Available Service Points attribute 474	AmDistributionList (C++) 205	Message Content Descriptor tag	
	AmDistributionList (Java) 403	names 510	
D	AmMessage (C++) 191	message types 510	
В	AmMessage (Java) 390	publish/subscribe 511	
base classes	AmObject (C++) 216	reason codes 512	
C++ 149, 183	AmObject (Java) 413	service and policy definitions 509	
Java 355, 383	AmPolicy (C++) 212	string length 510	
begin	AmPolicy (Java) 410	system default names and	
AmSession (C++) 186	AmPublisher (C++) 207	handles 515	
AmSession (Java) 386	AmPublisher (Java) 405	wait time 515	
bibliography 521	AmReceiver (C++) 202	content-based publish/subscribe 459	
Bind On Open attribute 476	AmReceiver (Java) 400	Convert attribute 478	
BookManager 523	AmSender (C++) 198	createDistributionList	
Boolean constants 509	AmSender (Java) 396	AmSession (C++) 186	
Broker Service attribute 474	AmSession (C++) 186	AmSession (Java) 386	
browse	AmSession (Java) 386	createMessage	
AmReceiver (C++) 201	AmSubscriber (C++) 209	AmSession (C++) 187	
AmReceiver (Java) 399	AmSubscriber (Java) 407	AmSession (Java) 387	
constants 509	close	createPolicy	
building applications	AmDistributionList (C++) 205	AmSession (C++) 187	
C 29	AmDistributionList (Java) 403	AmSession (Java) 387	
C++ 161	AmPublisher (C++) 207	createPublisher	
COBOL 241	AmPublisher (Java) 405	AmSession (C++) 187	
Java 366	AmReceiver (C++) 202	AmSession (Java) 387	
	AmReceiver (Java) 400	createReceiver	
	AmSender (C++) 198 AmSender (Java) 396	AmSession (C++) 187	
C	AmSender (Java) 396 AmSession (C++) 186	AmSession (Java) 387	
	AmSession (G++) 180 AmSession (Java) 386	createSender	
C++ applications	AmSubscriber (C++) 209	AmSession (C++) 187	
AIX 161	AmSubscriber (G++) 209 AmSubscriber (Java) 407	AmSession (Java) 387	
AS/400 162	- ·	createSession	
HP-UX 163	closing a session C 16	AmSessionFactory (C++) 184	
Solaris 164	C++ 151	AmSessionFactory (Java) 384	
Windows 165	COBOL 230	createSubscriber	
C++ interface	Java 357	AmSession (C++) 187	
overview 167	COBOL applications	AmSession (Java) 387	
reference information 183	OS/390 241	creating MQSeries objects 460	
using 149	COBOL high-level interface	creating objects	
C applications	equivalent object interface	C++ 151	
AIX 30	functions 280	Java 357	
AS/400 31	overview 244		
HP-UX 32	reference information 246	_	
OS/390 33	using 227	D	
Solaris 34	COBOL object interface	data types, C 29	
Windows 35	overview 269	datagram	
C high-level interface	reference information 283	C 16	
equivalent object interface	using 227	C++ 152	
functions 74	9	C++ 152 COBOL 230	
overview 38	commit	Java 357	
reference information 39	AmSession (C++) 186		
using 13	AmSession (Java) 386	Default MCD Domain attribute 472	
C object interface	compilers 426	Default MCD Format attribute 472	
overview 63	completion code constants 509	Default MCD Format attribute 472	
reference information 77	connecting to MQSeries 458	Default MCD Set attribute 472	
using 13	Connection Mode attribute 475	Default MCD Type attribute 472	

default objects	distribution list interface (C) (continued)	Field Disposition attribute 478
C 14	amDstOpen 128	field limits
C++ 150	amDstSend 129	C 28
COBOL 228	amDstSendFile 130	C++ 160
Java 356	distribution list interface (COBOL)	COBOL 240
	, , , , , , , , , , , , , , , , , , , ,	
system 470	AMDLCL 332	Java 365
definition	AMDLCLEC 332	file transfer
distribution list 474	AMDLGTLE 332	C 21
policy 469, 475	AMDLGTNA 333	C++ 155
publisher 474	AMDLGTSC 333	COBOL 235
service 469	AMDLGTSH 334	Java 361
service point (sender/receiver) 472	AMDLOP 334	filters 459
subscriber 474	AMDLSN 335	filters for publish/subscribe 459
system provided 470	AMDLSNFL 335	format constants 510
Definition Type attribute 472	Dynamic Queue Prefix attribute 472	
definition type constants 509	Dynamic Queue Frenz attribute 4/2	
		G
Delete On Close attribute 478	E	getCCSID
deleteDistributionList	E	AmMessage (C++) 192
AmSession (C++) 188	elements, name/value	9 1
deleteElement	C 24	AmMessage (Java) 391
AmMessage (C++) 191	C++ 157	AmPublisher (C++) 207
AmMessage (Java) 390	COBOL 237	AmPublisher (Java) 405
deleteFilter		AmSender (C++) 198
AmMessage (C++) 191	Java 362	AmSender (Java) 396
AmMessage (Java) 390	enableWarnings	AmSubscriber (C++) 209
deleteMessage	AmDistributionList (C++) 205	AmSubscriber (Java) 407
AmSession (C++) 188	AmDistributionList (Java) 403	getClassName
, ,	AmMessage (C++) 192	AmErrorException (C++) 221
deleteNamedElement	AmMessage (Java) 391	AmErrorException (Java) 416
AmMessage (C++) 192	AmPolicy (C++) 212	AmException (C++) 220
AmMessage (Java) 390	AmPolicy (Java) 410	AmException (Java) 415
deletePolicy	AmPublisher (C++) 207	*
AmSession (C++) 188	AmPublisher (Java) 405	AmWarningException (C++) 222
deletePublisher	AmReceiver (C++) 202	AmWarningException (Java) 417
AmSession (C++) 188	AmReceiver (Java) 400	getCompletionCode
deleteReceiver		AmErrorException (C++) 221
AmSession (C++) 188	AmSender (C++) 198	AmErrorException (Java) 416
deleteSender	AmSender (Java) 396	AmException (C++) 220
AmSession (C++) 188	AmSession (C++) 188	AmException (Java) 415
deleteSession	AmSession (Java) 388	AmStatus (C++) 217
	AmSubscriber (C++) 209	AmStatus (Java) 414
AmSessionFactory (C++) 184	AmSubscriber (Java) 407	AmWarningException (C++) 222
deleteSubscriber	Encoding attribute 472	AmWarningException (Java) 417
AmSession (C++) 188	encoding constants 510	getCorrelationId
deleteTopic	error handling	8
AmMessage (C++) 192	C 25	AmMessage (C++) 192
AmMessage (Java) 391	C++ 157	AmMessage (Java) 391
deleting C++ objects 152	COBOL 239	getDataLength
directory structure	Java 363	AmMessage (C++) 192
AIX 430		AmMessage (Java) 391
	examples	getDataOffset
AS/400 433	C 16	AmMessage (C++) 192
HP-UX 438	C++ 151	AmMessage (Java) 391
OS/390 441	COBOL 230	getDefinitionType
Solaris 445	Java 357	AmReceiver (C++) 202
Windows 448	Exception Action attribute 476	AmReceiver (Java) 400
disk space 425	exception classes	AmSubscriber (C++) 209
distribution list definition 474	C++ 150, 183	AmSubscriber (Java) 407
distribution list interface	Java 356, 383	- ,
overview (C) 70	Expiry Interval attribute 476	getElement
. ,	1 7	AmMessage (C++) 193
overview (C++) 175		AmMessage (Java) 391
overview (COBOL) 276	F	getElementCCSID
overview (Java) 376	-	AmMessage (C++) 193
distribution list interface (C)	failure (of AMI program)	getElementCount
amDstClearErrorCodes 126	common causes 493	AmMessage (C++) 193
amDstClose 126	reason codes 492	AmMessage (Java) 391
amDstGetLastError 127	symptom report (OS/390) 492	getEncoding
amDstGetName 127	symptom report (UNIX and	AmMessage (C++) 193
amDstGetSenderCount 128	Windows) 492	AmMessage (Java) 392
amDstGetSenderHandle 128	feedback codes 510	AmPublisher (C++) 207
	TOURDHER COULD DID	111111 (101101101 (1011) 207

(T 1: (ti1)		- T
getEncoding (continued)	getName (continued)	getType
AmPublisher (Java) 405	AmReceiver (Java) 401	AmMessage (C++) 195
AmSender (C++) 198	AmSender (C++) 199	AmMessage (Java) 394
AmSender (Java) 396	AmSender (Java) 397	getValue
AmSubscriber (C++) 209	AmSession (C++) 189	AmElement (C++) 215
AmSubscriber (Java) 407	AmSession (Java) 388	AmElement (Java) 412
getFactoryName	AmSubscriber (C++) 210	getVersion
AmSessionFactory (C++) 184	AmSubscriber (Java) 408	AmElement (C++) 215
AmSessionFactory (Java) 384	getNamedElement	AmElement (Java) 412
getFilter	AmMessage (C++) 194	getWaitTime
AmMessage (C++) 193	AmMessage (Java) 393	AmPolicy (C++) 212
AmMessage (Java) 392	getNamedElementCount	AmPolicy (Java) 410
getFilterCount	AmMessage (C++) 194	glossary 519
AmMessage (C++) 193	AmMessage (Java) 393	group status constants 510
AmMessage (Java) 392	getQueueName	0 1
getFormat		ш
	AmReceiver (C++) 203	Н
0 \ /	AmReceiver (Java) 401	handle constants 510
AmMessage (Java) 392	AmSubscriber (C++) 210	Handle Poison Message attribute 478
getGroupStatus	AmSubscriber (Java) 408	header file
AmMessage (C++) 194	getReasonCode	
AmMessage (Java) 392	AmErrorException (C++) 221	C 29
getLastErrorStatus	AmErrorException (Java) 416	C++ 161
AmDistributionList (C++) 205		helper classes
	AmException (C++) 220	C++ 150, 183
9 /	AmException (Java) 415	Java 356, 383
AmMessage (C++) 194	AmStatus (C++) 217	helper macros 108
AmMessage (Java) 392	AmStatus (Java) 414	high-level interface
AmObject (C++) 216	AmWarningException (C++) 222	equivalent object interface
AmObject (Java) 413	AmWarningException (Java) 417	functions 74
AmPolicy (C++) 212	getReasonCode2	
AmPolicy (Java) 410	AmStatus (C++) 217	using 13
AmPublisher (C++) 207		high-level interface (C)
AmPublisher (Java) 405	AmStatus (Java) 414	amBackout 40
AmReceiver (C++) 203	getReportCode	amBegin 41
	AmMessage (C++) 195	amBrowseMsg 42
AmReceiver (Java) 401	AmMessage (Java) 393	amCommit 44
AmSender (C++) 199	getRepository	amInitialize 45
AmSender (Java) 397	AmSessionFactory (C++) 184	amPublish 46
AmSession (C++) 189	AmSessionFactory (Java) 384	amReceiveFile 47
AmSession (Java) 388		amReceiveMsg 49
AmSubscriber (C++) 209	getSender	O O
AmSubscriber (Java) 407	AmDistributionList (C++) 205	amReceivePublication 51
getLocalHost	AmDistributionList (Java) 403	amReceiveRequest 53
AmSessionFactory (C++) 184	getSenderCount	amSendFile 55
	AmDistributionList (C++) 205	amSendMsg 56
AmSessionFactory (Java) 384	AmDistributionList (Java) 403	amSendRequest 57
getMessageId	getSource	amSendResponse 58
AmMessage (C++) 194	AmErrorException (C++) 221	amSubscribe 59
AmMessage (Java) 393		amTerminate 60
getMethodName	AmErrorException (Java) 416	amUnsubscribe 61
AmErrorException (C++) 221	AmException (C++) 220	overview 38
AmErrorException (Java) 416	AmException (Java) 415	reference information 39
AmException (C++) 220	AmWarningException (C++) 222	
	AmWarningException (Java) 417	high-level interface (COBOL)
AmException (Java) 415	getTopic	AMHBACK 247
AmWarningException (C++) 222	AmMessage (C++) 195	AMHBEGIN 248
AmWarningException (Java) 417	AmMessage (Java) 393	AMHBRMS 249
getName	_	AMHCMIT 251
AmDistributionList (C++) 205	getTopicCount	AMHINIT 252
AmDistributionList (Java) 403	AmMessage (C++) 195	AMHPB 253
AmElement (C++) 215	AmMessage (Java) 393	AMHRCFL 254
AmElement (Java) 412	getTraceLevel	AMHRCMS 256
,	AmSession (C++) 189	
AmMessage (C++) 194	AmSession (Java) 388	AMHRCPB 258
AmMessage (Java) 393	- ,	AMHRCRQ 260
AmObject (C++) 216	AmSessionFactory (C++) 184	AMHSB 266
AmObject (Java) 413	AmSessionFactory (Java) 384	AMHSNFL 262
AmPolicy (C++) 212	getTraceLocation	AMHSNMS 263
AmPolicy (Java) 410	AmSession (C++) 189	AMHSNRQ 264
AmPublisher (C++) 207	AmSession (Java) 388	AMHSNRS 265
AmPublisher (Java) 405	AmSessionFactory (C++) 184	AMHTERM 267
AmReceiver (C++) 203	AmSessionFactory (Java) 384	AMHUN 268

high-level interface (COBOL) (continued) equivalent object interface functions 280	M macros, helper 108	message interface (COBOL) (continued) AMMSGTDO 302 AMMSGTEC 303
overview 244	Message Content Descriptor tag	AMMSGTEL 303
reference information 246	names 510 Message Context attribute 476	AMMSGTEN 304
using 227 HP-UX	message interface	AMMSGTFC 304 AMMSGTFI 305
C++ applications 163	overview (C) 66	AMMSGTFO 305
C applications 32	overview (COPOL) 272	AMMSGTGS 306
installation 436	overview (COBOL) 272 overview (Java) 372	AMMSGTLE 306
Java applications 366	message interface (C)	AMMSGTMI 307 AMMSGTNA 307
prerequisites 425 HTML (Hypertext Markup	amMsgAddElement 91	AMMSGTNC 308
Language) 522	amMsgAddFilter 91	AMMSGTNE 308
Hypertext Markup Language	AmMsgAddStreamName 108 amMsgAddTopic 92	AMMSGTRC 309
(HTML) 522	amMsgClearErrorCodes 92	AMMSGTTC 310
	amMsgDeleteElement 92	AMMSGTTO 309 AMMSGTTY 310
	amMsgDeleteFilter 93	AMMSREBY 311
I	amMsgDeleteNamedElement 93	AMMSRS 311
Implicit Open attribute 476, 478	amMsgDeleteTopic 94 amMsgGetCCSID 94	AMMSSELC 313
include file	amMsgGetCorrelId 94	AMMSSTCC 312
C 29	amMsgGetDataLength 95	AMMSSTCI 312 AMMSSTDO 312
C++ 161	amMsgGetDataOffset 95	AMMSSTEN 313
Inform If Retained attribute 479 initial values for structures 29	amMsgGetElement 95	AMMSSTFO 314
installation	amMsgGetElementCCSID 96 amMsgGetElementCount 96	AMMSSTGS 314
administration tool 457	amMsgGetEncoding 96	AMMSWRBY 315
AIX 428	amMsgGetFilter 97	message types 510 messages 4
AS/400 432	amMsgGetFilterCount 97	messages, poison 478
HP-UX 436	amMsgGetFormat 98	messages, publish/subscribe
OS/390 440 prerequisites 425	amMsgGetGroupStatus 98 amMsgGetLastError 99	C 22
Solaris 443	amMsgGetMsgId 99	C++ 156
Windows 447	amMsgGetName 100	COBOL 235 Java 361
interface	amMsgGetNamedElement 100	messages, receiving
C++ 167, 183	amMsgGetNamedElementCount 101	C 18
C high-level 37, 38	AmMsgGetPubTimeStamp 108 amMsgGetReportCode 101	C++ 153
C object 63, 77 COBOL high-level 243, 244	AmMsgGetStreamName 109	COBOL 232 Java 359
COBOL high-level 243, 244 COBOL object 269, 283	amMsgGetTopic 102	messages, request/response
Java 369, 383	amMsgGetTopicCount 103	C 19
interoperability 3	amMsgGetType 102	C++ 154
	amMsgReadBytes 103 amMsgReset 103	COBOL 234
	amMsgSetCCSID 104	Java 360
J	amMsgSetCorrelId 104	messages, sending C 16
jar file (Java) 366	amMsgSetDataOffset 104	C++ 152
Java applications	amMsgSetElementCCSID 105 amMsgSetEncoding 105	COBOL 230
AIX 366	amMsgSetFlooding 105	Java 357
AS/400 366	amMsgSetGroupStatus 106	model of the AMI 7
HP-UX 366 Solaris 366	amMsgWriteBytes 107	Model Queue Name attribute 472 MQSeries client
Windows 366	helper macros 108	connecting to 458
Java interface	message interface (COBOL) AMMSADEL 298	prerequisites 426
overview 369	AMMSADFI 298	MQSeries environment 426
reference information 383	AMMSADTO 299	MQSeries function calls
using 355	AMMSCLEC 299	C 27
	AMMSDEEL 299	C++ 159 COBOL 240
1	AMMSDEFI 300 AMMSDENE 300	Java 365
	AMMSDETO 301	MQSeries Integrator V2 472
Leave Queue Open attribute 476, 478	AMMSGELC 301	Migrating API applications to 459
local host cache (OS/390) 454 local host file 450	AMMSGTCC 301	Using the AMI with 458
local host file (OS/390) 453	AMMSGTCI 302	MQSeries Integrator Version 1,
10cai 1105t 111e (03/370) 433	AMMSGTDL 302	using 458

MQSeries objects, creating 460	OS/390 (continued)	publish
MQSeries publications 521	COBOL applications 241	AmPublisher (C++) 208
MQSeries Publish/Subscribe 458	installation 440	AmPublisher (Java) 406
MQSeries server	prerequisites 425	Publish Locally attribute 480
connecting to 458	OS/390 subsystems, application	Publish On Request Only attribute 479
prerequisites 426	advice 421	publish/subscribe
multithreading	overloading	constants 511
C 27	C++ 151	content-based 459
C++ 159	Java 357	filters 459
COBOL 240	overview	introduction 5
Java 365	C++ interface 167	using 458
	C high-level interface 38	publish/subscribe messaging
	C object interface 63	C 22
N	COBOL high-level interface 244	C++ 156
	COBOL object interface 269	COBOL 235
Name attribute	Java interface 369	Java 361
distribution list 474	•	Publish To Others Only attribute 480
policy 475		publisher definition 474
publisher 474	Р	publisher interface
service point 472	Г	overview (C) 71
name/value elements	PDF (Portable Document Format) 522	
C 24	Persistence attribute 476	overview (C++) 176
C++ 157	point-to-point 5	overview (COBOL) 277
COBOL 237	poison messages 478	overview (Java) 377
Java 362	-	publisher interface (C)
New Publications Only attribute 479	policy	amPubClearErrorCodes 131
New 1 dolleations Only attribute 4/7	constants 509	amPubClose 131
0	defining 469	amPubGetCCSID 131
0	general attributes 476	amPubGetEncoding 132
OAMAS subset 28	initialization attributes 475	amPubGetLastError 132
oamasami.h header 28	publish attributes 480	amPubGetName 133
object interface	receive attributes 478	amPubOpen 133
overview 63	send attributes 476	amPubPublish 134
reference information 77	subscribe attributes 479	publisher interface (COBOL)
object interface (COBOL)	summary 6	
overview 269	policy interface	AMPBCL 337
reference information 283		AMPBCLEC 337
object-style interface 13	overview (C) 73	AMPBGTCC 337
	overview (C++) 178	AMPBGTEN 338
object-style interface (COBOL) 227	overview (COBOL) 279	AMPBGTLE 338
objects	overview (Java) 379	AMPBGTNA 339
C 13	policy interface (C)	AMPBOP 339
C++ 149	amPolClearErrorCodes 142	AMPBPB 340
COBOL 227	amPolGetLastError 142	
Java 355	amPolGetName 143	
open	amPolGetWaitTime 143	Q
AmDistributionList (C++) 205	amPolSetWaitTime 143	Q
AmDistributionList (Java) 403	policy interface (COBOL)	Queue Manager Name attribute 472
AmPublisher (C++) 208	AMPOCLEC 348	Queue Name attribute 472
AmPublisher (Java) 406		D
AmReceiver (C++) 203	AMPOGTLE 348	R
AmReceiver (Java) 401	AMPOGTNA 349	readBytes
AmSender (C++) 199	AMPOGTWT 349	AmMessage (C++) 195
AmSender (Java) 397	AMPOSTWT 350	AmMessage (Java) 394
AmSession (C++) 189	Portable Document Format (PDF) 522	reason codes
AmSession (Java) 388	PostScript format 523	constants 512
	prerequisites	
AmSubscriber (C++) 210	compilers 426	description 497
AmSubscriber (Java) 408	disk space 425	receive
Open Shared attribute 478	MQSeries environment 426	AmReceiver (C++) 203, 204, 402
opening a session	OAMAS subset 28	AmReceiver (Java) 401
C 16	operating systems 425	AmSubscriber (C++) 210
C++ 151		AmSubscriber (Java) 408
COBOL 230	Priority attribute 476	receiver definition 472
Java 357	problem determination 481	receiver interface
opening objects	problems, causes of 493	overview (C) 69
C++ 151	procedural interface 13	overview (C++) 174
Java 357	procedural interface (COBOL) 227	overview (COBOL) 275
operating systems 425	programming languages 4	overview (Java) 375
OS/390	publications	receiver interface (C)
C applications 33	MQSeries 521	amRcvBrowse 115
11	-	

receiver interface (C) (continued)	Retry Interval attribute 476	sending messages
amRcvBrowseSelect 117	RF Header 472	C 16
amRcvClearErrorCodes 118	rollback	C++ 152
amRcvClose 119	AmSession (C++) 189	COBOL 230
amRcvGetDefnType 119	AmSession (Java) 388	Java 357
amRcvGetLastError 120	runtime environment	service
amRcvGetName 120	AIX 429	constants 509
amRcvGetQueueName 121	AS/400 432	defining 469
amRcvOpen 121	HP-UX 437	summary 4
amRcvReceive 122	OS/390 440	service point 472
amRcvReceiveFile 124	Solaris 444	
amRcvSetQueueName 125	Windows 447	Service Type attribute 472
receiver interface (COBOL)	Timas Ti	session factory
AMRCBR 322		overview (C++) 168
AMRCBRSE 324	S	overview (Java) 370
AMRCCL 326		session interface
AMRCCLEC 325	sample programs	overview (C) 64
AMRCGTDT 326	AS/400 461	overview (C++) 169
AMRCGTLE 327	OS/390 464	overview (COBOL) 270
AMRCGTNA 327	UNIX 461	overview (Java) 371
AMRCGTQN 328	Windows 461	session interface (C)
AMRCOP 328	Segmentation attribute 476	amSesBegin 78
	send	amSesClearErrorCodes 78
AMRCRC 329	AmDistributionList (C++) 206	amSesClose 79
AMRCRCFL 330	AmDistributionList (Java) 404	amSesCommit 79
AMRCSTQN 331	AmSender (C++) 199	amSesCreate 79
Receiver Service attribute 474	AmSender (Java) 397	amSesCreateDistList 80
receiving files	sender definition 472	amSesCreateMessage 80
C 21	sender interface	amSesCreatePolicy 80
C++ 155	overview (C) 68	amSesCreatePublisher 81
COBOL 235	overview (C++) 173	amSesCreateReceiver 81
Java 361	overview (COBOL) 274	amSesCreateSender 82
receiving messages	overview (Java) 374	amSesCreateSubscriber 82
C 18	sender interface (C)	amSesDelete 83
C++ 153	amSndClearErrorCodes 110	amSesDeleteDistList 83
COBOL 232	amSndClose 110	amSesDeleteMessage 83
Java 359	amSndGetCCSID 111	amSesDeletePolicy 84
reference information	amSndGetEncoding 111	amSesDeletePublisher 84
C++ interface 183	amSndGetLastError 111	amSesDeleteReceiver 84
C high-level interface 39	amSndGetName 112	amSesDeleteSender 85
C object interface 77	amSndOpen 112	amSesDeleteSubscriber 85
COBOL high-level interface 246	amSndSend 113	amSesGetDistListHandle 85
COBOL object interface 283	amSndSendFile 114	amSesGetDistEistHandle 83
Java interface 383	sender interface (COBOL)	
Report Data attribute 476		amSesGetMessageHandle 86
Report Type COA attribute 476	AMSNCL 317 AMSNCLEC 316	amSesGetPolicyHandle 87
Report Type COD attribute 476	AMSNGTCC 317	amSesGetPublisherHandle 87
Report Type Exception attribute 476		amSesGetReceiverHandle 87
Report Type Expiry attribute 476	AMSNGTEN 317	amSesGetSenderHandle 88
repository, using	AMSNGTLE 318	amSesGetSubscriberHandle 88
C 14	AMSNGTNA 318	amSesOpen 88
C++ 150	AMSNOP 319	amSesRollback 89
COBOL 228	AMSNSN 319	transaction coordination 78
Java 356	AMSNSNFL 320	unit of work 78
repository cache (OS/390) 454	sendFile	session interface (COBOL)
repository file 450	AmDistributionList (C++) 206	AMSEBG 284
	AmDistributionList (Java) 404	AMSECL 285
repository file (OS/390) 453	AmSender (C++) 200	AMSECLEC 284
request/response messaging	AmSender (Java) 398	AMSECM 285
C 19	sending files	AMSECR 286
C++ 154	C 21	AMSECRDL 286
COBOL 234	C++ 155	AMSECRMS 287
Java 360	COBOL 235	AMSECRPB 288
reset	Java 361	AMSECRPO 287
AmMessage (C++) 195	sending group messages	AMSECRRC 288
AmMessage (Java) 394	C 26	AMSECRSB 289
Response Correl Id attribute 476	C++ 159	AMSECRSN 289
Retain attribute 480	COBOL 240	AMSEDL 290
Retry Count attribute 476	Java 365	AMSEDLDL 290

session interface (COBOL) (continued)	Solaris (continued)	Т
AMSEDLMS 290	C applications 34	terminalacty used in this book 510
AMSEDLPB 291	installation 443	terminology used in this book 519 tool, administration 457
AMSEDLPO 291	Java applications 366	topics, publish/subscribe
AMSEDLRC 291	prerequisites 425	C 22
AMSEDLSB 292	Split File attribute 476	C++ 156
AMSEDLSN 292	string length constants 510	COBOL 235
AMSEGHDL 292	structure of the AMI	Java 361
AMSEGHMS 293	C 13	toString
AMSEGHPO 204	C++ 149	AmElement (C++) 215
AMSECHEC 204	COBOL 227	AmElement (Java) 412
AMSEGHRC 294	Java 355	AmErrorException (C++) 221
AMSEGHSB 295 AMSEGHSN 295	structure of this book xix	AmErrorException (Java) 416
AMSEGTLE 293	structures, initial values 29	AmException (C++) 220
AMSEGTEE 293 AMSEOP 296	subscribe	AmException (Java) 415
AMSERB 296		AmStatus (C++) 217
transaction coordination 284	AmSubscriber (C++) 211	AmStatus (Java) 414
unit of work 284	AmSubscriber (Java) 409	AmWarningException (C++) 222
setCCSID	content-based 459 filters 459	AmWarningException (Java) 417
AmMessage (C++) 196		trace
AmMessage (Java) 394	Subscribe Locally attribute 479	AS/400 482
setCorrelationId	subscriber definition 474	C++ and Java 484
AmMessage (C++) 196	subscriber interface	example 484
AmMessage (Java) 394	overview (C) 72	UNIX 482
setDataOffset	overview (C++) 177	using, OS/390 489
AmMessage (C++) 196	overview (COBOL) 278	using, UNIX and Windows 481
AmMessage (Java) 394	overview (Java) 378	Windows 483
setElementCCSID	subscriber interface (C)	transaction coordination
AmMessage (C++) 196	amSubClearErrorCodes 135	C 78
setEncoding	amSubClose 135	C++ 186
AmMessage (C++) 196	amSubGetCCSID 135	COBOL 284
AmMessage (Java) 395	amSubGetDefnType 136	Java 386
setFormat	amSubGetEncoding 136	transaction support
AmMessage (C++) 197	amSubGetLastError 137	C 26
AmMessage (Java) 395	amSubGetName 137	C++ 158
setGroupStatus	amSubGetQueueName 138	COBOL 240
AmMessage (C++) 197	amSubOpen 138	Java 364
AmMessage (Java) 395	amSubReceive 139	Trusted Option attribute 475
setLocalHost	amSubSetQueueName 139	
AmSessionFactory (C++) 185	amSubSubscribe 140	11
AmSessionFactory (Java) 384	amSubUnsubscribe 141	U
setQueueName	subscriber interface (COBOL)	Unicode character conversion 440
AmReceiver (C++) 204	AMSBCL 341	unit of work
AmReceiver (Java) 402	AMSBCLEC 341	C 26, 78
AmSubscriber (C++) 210	AMSBGTCC 342	C++ 158, 186
AmSubscriber (Java) 408	AMSBGTDT 342	COBOL 240, 284
setRepository	AMSBGTEN 343	Java 364, 386
AmSessionFactory (C++) 185	AMSBGTLE 343	UNIX
AmSessionFactory (Java) 385	AMSBGTNA 344	sample programs 461
setTraceLevel	AMSBGTQN 344	unsubscribe
AmSessionFactory (C++) 185	AMSBOP 345	AmSubscriber (C++) 211
AmSessionFactory (Java) 385	AMSBRC 345	AmSubscriber (Java) 409
setTraceLocation	AMSBSB 346	Unsubscribe All attribute 479
AmSessionFactory (C++) 185	AMSBSTQN 346	Use Correl Id As Id attribute 479, 480
AmSessionFactory (Java) 385	AMSBUN 347	using the AMI
setVersion	SupportPac 523	C 13
AmElement (C++) 215	Suppress Registration attribute 480	C++ 149
AmElement (Java) 412	Syncpoint attribute 476	COBOL 227
setWaitTime		Java 355
AmPolicy (C++) 212	system default handle synonyms 515	
AmPolicy (Java) 410	system default names 515	147
simulated group messages 26, 240	system default objects	W
Simulated Group Support attribute 472	C 14	Wait For Whole Group attribute 478
softcopy books 522	C++ 150	Wait Interval attribute 478
Solaris	COBOL 228	Wait Interval Read Only attribute 478
C++ applications 164	Java 356	wait time constants 515
÷ ÷		

what you need to know xix who this book is for xix Windows C++ applications 165 C applications 35 installation 447 Java applications 366 prerequisites 425 sample programs 461 Windows Help 523 writeBytes AmMessage (C++) 197 AmMessage (Java) 395 writing applications C 16 C++ 151 COBOL 230 Java 357 writing applications for OS/390subsystems 421 writing IMS applications 421

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 44–1962–870229
 - From within the U.K., use 01962-870229
- 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.

IBM.



Printed in the United States of America on recycled paper containing 10% recovered post-consumer fiber.

SC34-5604-05

