

Application Messaging Interface



Application Messaging Interface

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

Eighth edition (August 2001)

This edition applies to $IBM^{\tiny{\textcircled{\tiny B}}}$ MQSeries Application Messaging Interface Version 1.2, and to any subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1999, 2001. All rights reserved.

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

Contents

Figures xvii	C applications on AIX	. 30
5	C applications on AS/400	
Tables xix	C applications on HP-UX	
Tables	C applications on OS/390	
	C applications on Solaris	
About this book xxi	C applications on Windows	
Who this book is for xxi	Writing policy handlers in C	
What you need to know to understand this book xxi	Compiling, linking and installing a policy	
Structure of this book xxi	handler	. 36
Conventions used in this book xxii		
	Chapter 3. The C high-level interface	41
Summary of changes xxiii	Overview of the C high-level interface	
Changes for this edition (SC34-5604-07) xxiii	Initialize and terminate	
Changes for the seventh edition (SC34-5604-06) xxiii	Sending messages	
Changes for the sixth edition (SC34-5604-05) xxiv	Receiving messages	
` , , , , , , , , , , , , , , , , , , ,	File transfer	
Part 1. Introduction	Publish/subscribe	
Part I. Introduction	Transaction support	
	Reference information for the C high-level interface	43
Chapter 1. Introduction 3	amBackout	
Main components of the AMI	amBegin	
Sending and receiving messages	amBrowseMsg	
Interoperability	amCommit	. 40
Programming languages 4	amInitialize	. 40
Description of the AMI 4	amPublish	
Messages 4		
Services	amReceiveFile	
Policies 6	amReceiveMsg	. 50
Application Messaging Interface model 7		
Further information 8	amReceiveRequest	. 37
	amSendMsg	. 5:
Part 2. The C interface 9	amSendRequest	
Tart 2. The O interface	amSendResponse	
	amSubscribe	. 62
Chapter 2. Using the Application	amTerminate	. 6/
Messaging Interface in C 13	amUnsubscribe	
Structure of the AMI	amonsubscribe	. 00
Using the repository	Chapter 4 C chiest interfess everyion	67
System default objects	Chapter 4. C object interface overview	
Writing applications in C	Session interface functions	
Opening and closing a session 16	Session management	
Sending messages	Create objects	. 68
Receiving messages	Get object handles	. 68
Request/response messaging	Delete objects	. 69
File transfer	Transactional processing	
Publish/subscribe messaging	Error handling	
Using name/value elements 24	Message interface functions	
Error handling	Get values	
Transaction support 26	Set values	
Sending group messages 26	Reset values	
Other considerations	Read and write data	
Using the AMI OAMAS subset 28	Publish/subscribe topics	
Building C applications	Publish/subscribe filters	
AMI include file	Publish/subscribe name/value elements	
Data types	Error handling	. 71
Initial values for structures	Publish/subscribe helper macros	. 71

Open and close 72 amsesCoSubscriberHandle Send. 72 amsesOpen Get values. 72 amsesOpen Error handling 72 Message interface functions A Copen and close 73 ammsesCellblack Receiver and browse 73 ammsgAddFilter Get values. 73 ammsgAddTopic Berror handling 73 ammsgSpeleteFlament Bror handling 73 ammsgSpeleteFlament Bror handling 74 ammsgSpeleteFlament Bror handling 74 ammsgSpeleteNamedElement Send 74 ammsgSpeleteNamedElement A Copen and close 74 ammsgSpeleteNamedElement A Copen and close 75 ammsgCettlament A Copen and close 75 ammsgCettlateIment A Copen and close 75 ammsgCettlateIment <tr< th=""><th>Open and close 72 am%seGelsubscriberHandle 92 Get values 72 am%seSpene 92 Error handling 72 messege interface functions 93 Receiver interface functions 73 amMsgAddElement 99 Open and close 73 amMsgAddFilter 99 Receive and browse 73 amMsgAddFilter 99 Get values 73 amMsgDeleteFlement 99 Ser values 73 amMsgDeleteFlement 99 Berror handling 73 amMsgDeleteFlement 99 Send. 74 amMsgDeleteFlement 99 Send. 74 amMsgGetCSDD 99 Send. 74 amMsgGetCSDD 99 Send. 74 amMsgGetDataLength 99 Publish 75 amMsgGetDataLength 99 Publish 75 amMsgGetDataLength 99 Subscriber interface functions 75 amMsgGetElement 90 Ser values 76</th><th></th><th></th><th></th></tr<>	Open and close 72 am%seGelsubscriberHandle 92 Get values 72 am%seSpene 92 Error handling 72 messege interface functions 93 Receiver interface functions 73 amMsgAddElement 99 Open and close 73 amMsgAddFilter 99 Receive and browse 73 amMsgAddFilter 99 Get values 73 amMsgDeleteFlement 99 Ser values 73 amMsgDeleteFlement 99 Berror handling 73 amMsgDeleteFlement 99 Send. 74 amMsgDeleteFlement 99 Send. 74 amMsgGetCSDD 99 Send. 74 amMsgGetCSDD 99 Send. 74 amMsgGetDataLength 99 Publish 75 amMsgGetDataLength 99 Publish 75 amMsgGetDataLength 99 Subscriber interface functions 75 amMsgGetElement 90 Ser values 76			
Send. 72 amSesOpen Get values. 72 amSesRollback Receiver interface functions 73 amMsgAddFlement A Receive and browse 73 amMsgAddFlement Get values. 73 amMsgAddFlement Error handling 73 amMsgDeleteFlier Distribution list interface functions 74 amMsgDeleteFlier Open and close 74 amMsgDeleteFlier Get values. 74 amMsgCetDetTopic c Get values. 74 amMsgCetDataLength Error handling 74 amMsgCetDataLength Publisher interface functions. 75 amMsgCetDataLength Open and close 75 amMsgCetElementCoSID amMsgCetElementCoSID Bror handling 75 amMsgCetElementCosID amMsgCetElementCosII Get values 75 amMsgCetElementCosII amMsgCetElementCosII Bror handling 76 amMsgCetElementCosII amMsgCetElementCosII Get values 76 amMsgCetElementCosII amMsgCetIntercont Set valu	Send. 72 amSesOpen 95 Get values 72 mSesRollback 95 Error handling 72 Message interface functions 95 Open and close 73 amMsgAddFilter 99 Receive and browse 73 amMsgAddTopic 99 Get values 73 amMsgAddTopic 99 Set values 73 amMsgDeleteEllment 99 Error handling 73 amMsgDeleteFilter 99 Distribution list interface functions 74 amMsgDeleteFilter 99 Den and close 74 amMsgGettCSID 99 Get values 74 amMsgGettDataOffset 99 Publish 75 amMsgGettDataOffset 99 Publish 75 amMsgGettElementCorpil 90 Open and close 75 amMsgGettElementCount 100 Get values 75 amMsgGettElementCount 100 Subscriber interface functions 76 amMsgGettElementCount 100 S			amSesGetSenderHandle
Send. 72 amSesOpen Get values. 72 amSesRollback Receiver interface functions 73 amMsgAddFlement A Receive and browse 73 amMsgAddFlement Get values. 73 amMsgAddFlement Error handling 73 amMsgDeleteFlier Distribution list interface functions 74 amMsgDeleteFlier Open and close 74 amMsgDeleteFlier Get values. 74 amMsgCetDetTopic c Get values. 74 amMsgCetDataLength Error handling 74 amMsgCetDataLength Publisher interface functions. 75 amMsgCetDataLength Open and close 75 amMsgCetElementCoSID amMsgCetElementCoSID Bror handling 75 amMsgCetElementCosID amMsgCetElementCosII Get values 75 amMsgCetElementCosII amMsgCetElementCosII Bror handling 76 amMsgCetElementCosII amMsgCetElementCosII Get values 76 amMsgCetElementCosII amMsgCetIntercont Set valu	Send. 72 amSesOpen 95 Get values 72 mSesRollback 95 Error handling 72 Message interface functions 95 Open and close 73 amMsgAddFilter 99 Receive and browse 73 amMsgAddTopic 99 Get values 73 amMsgAddTopic 99 Set values 73 amMsgDeleteEllment 99 Error handling 73 amMsgDeleteFilter 99 Distribution list interface functions 74 amMsgDeleteFilter 99 Den and close 74 amMsgGettCSID 99 Get values 74 amMsgGettDataOffset 99 Publish 75 amMsgGettDataOffset 99 Publish 75 amMsgGettElementCorpil 90 Open and close 75 amMsgGettElementCount 100 Get values 75 amMsgGettElementCount 100 Subscriber interface functions 76 amMsgGettElementCount 100 S	Open and close	. 72	amSesGetSubscriberHandle 92
Error handling	Error handling	Send	. 72	
Receive interface functions 73 amMsgAddEllement 73 cet values 73 amMsgDeleteFlement 74 amMsgDeleteFliller 75 amMsgDeleteFliller 75 amMsgDeleteFliller 76 amMsgDeleteFliller 76 amMsgDeleteFliller 77 amMsgDeleteFliller 78 amMsgDeleteFliller 79 amMsgGetCSSID 79 amMsgGetDataOffset 79 amMsgGetDataOffset 79 amMsgGetDataOffset 79 amMsgGetBlaementCSSID 79 amMsgGetBlaement 79 amMsgGetBlaeme	Receive interface functions	Get values	. 72	amSesRollback
Open and close 73 amMsgAddFilter Receive and browse 73 amMsgDeletFilter Get values 73 amMsgDeleteFilter Error handling 73 amMsgDeleteFilter Distribution list interface functions 74 amMsgDeleteFilter Open and close 74 amMsgDeleteFilter Send. 74 amMsgGetCSCID Get values 74 amMsgGetDataLength Error handling 74 amMsgGetDataLength Publisher interface functions 75 amMsgGetBlement Open and close 75 amMsgGetBlement 75 amMsgGetBlement amMsgGetBlement 80 75 amMsgGetBlement 9 amMsgGetBlement amMsgGetBlement 9 amMsgGetBlement amMsgGetBlement 9 amMsgGetBlementCoult amMsgGetBlementCoult 9 amMsgGetBlementCount amMsgGetBlementCount 9 amMsgGetTopicCount amMsgGetNamedElement 9 amMsgGetNamedElement amMsgGetNamedElement	Open and close 73 amMsgAddFliter 99 Get values 73 amMsgAddTopic 99 Get values 73 amMsgDeleteTeller 99 Set values 73 amMsgDeleteFilter 99 Error handling 73 amMsgDeleteFilter 99 Open and close 74 amMsgDeleteFilter 99 Send. 74 amMsgGetCSCID 99 Get values. 74 amMsgGetDatalength 99 Publisher interface functions. 75 amMsgGetElanent 99 Open and close 75 amMsgGetElement 99 Publish. 75 amMsgGetElementCSID 100 Get values. 75 amMsgGetElementCoxID 100 Get values. 75 amMsgGetElementCoxID 100 Subscriber interface functions 76 amMsgGetElementCoxID 100 Subscriber interface functions 76 amMsgGetElementCoxID 100 Get values. 76 amMsgGetFilterCount 100			Message interface functions
Receive and browse 73 amMsgClearErrorCodes Get values 73 amMsgClearErrorCodes Set values 73 amMsgDeleteFliter Error handling 73 amMsgDeleteFliter Distribution list interface functions 74 amMsgDeleteFliter Open and close 74 amMsgCetCatalength Error handling 74 amMsgCetDatalength Publisher interface functions 75 amMsgCetElement Open and close 75 amMsgCetElementCCSID Publish 75 amMsgCetElementCCSID Get values 75 amMsgCetElementCOIN Get values 75 amMsgCetElementCOIN Subscriber interface functions 76 amMsgCetFliter Open and close 76 amMsgCetFlormot Broker messages 76 amMsgCetFlormot Get values 76 amMsgCetFlormot Set value 76 amMsgCetGetGroupStatus Set value 77 amMsgCetNamedElement 1 Get values 77 amMsgCetNam	Receive and browse 73 amMsgCelarErrorCodes 99 Get values 73 amMsgDeleteEllement 99 Set values 73 amMsgDeleteEllement 99 Error handling 73 amMsgDeleteTilter 99 Open and close 74 amMsgDeleteTopic 99 Send. 74 amMsgDeleteTopic 99 Get values 74 amMsgDeleteTopic 99 Get values 74 amMsgDeleteTopic 99 Fublisher interface functions 75 amMsgCetDataLength 99 Publisher interface functions 75 amMsgCetDataLength 99 Publisher interface functions 75 amMsgCetDataLength 99 Publisher interface functions 75 amMsgCetEllement 99 Publisher interface functions 75 amMsgCetEllementCOSID 100 Get values 75 amMsgCetEllementCOSID 100 Get values 76 amMsgCetFilterCount 100 Set value 77 amMsgCetFoptatus			amMsgAddElement 95
Receive and browse 73 amMsgClearErrorCodes Get values 73 amMsgClearErrorCodes Set values 73 amMsgDeleteFliter Error handling 73 amMsgDeleteFliter Distribution list interface functions 74 amMsgDeleteFliter Open and close 74 amMsgCetCatalength Error handling 74 amMsgCetDatalength Publisher interface functions 75 amMsgCetElement Open and close 75 amMsgCetElementCCSID Publish 75 amMsgCetElementCCSID Get values 75 amMsgCetElementCOIN Get values 75 amMsgCetElementCOIN Subscriber interface functions 76 amMsgCetFliter Open and close 76 amMsgCetFlormot Broker messages 76 amMsgCetFlormot Get values 76 amMsgCetFlormot Set value 76 amMsgCetGetGroupStatus Set value 77 amMsgCetNamedElement 1 Get values 77 amMsgCetNam	Receive and browse 73 amMsgCelarErrorCodes 99 Get values 73 amMsgDeleteEllement 99 Set values 73 amMsgDeleteEllement 99 Error handling 73 amMsgDeleteTilter 99 Open and close 74 amMsgDeleteTopic 99 Send. 74 amMsgDeleteTopic 99 Get values 74 amMsgDeleteTopic 99 Get values 74 amMsgDeleteTopic 99 Fublisher interface functions 75 amMsgCetDataLength 99 Publisher interface functions 75 amMsgCetDataLength 99 Publisher interface functions 75 amMsgCetDataLength 99 Publisher interface functions 75 amMsgCetEllement 99 Publisher interface functions 75 amMsgCetEllementCOSID 100 Get values 75 amMsgCetEllementCOSID 100 Get values 76 amMsgCetFilterCount 100 Set value 77 amMsgCetFoptatus	Open and close	. 73	amMsgAddFilter 95
Get values 73 amMsgDeleteElement Set values 73 amMsgDeleteElement Error handling 73 amMsgDeleteFilter Distribution list interface functions 74 amMsgDeleteTopic Get values 74 amMsgDeleteTopic Get values 74 amMsgCetCSID Get values 74 amMsgCetDataLength Publisher interface functions 75 amMsgCetElementCSID Open and close 75 amMsgCetElementCSID Get values 75 amMsgCetElementCount Bror handling 75 amMsgCetElementCSID Set values 76 amMsgCetElementCount Bror handling 76 amMsgCetElementCount Get values 76 amMsgCetFlaterCount Set value 76 amMsgCetFlaterCount Bror handling 76 amMsgCetTopus Ferror handling 77 amMsgCetNamedElement Policy interface functions 78 amMsgCetRoproleot Set value 77 amMsgCetPloster	Get values 73 amMsgDeleteElment 99 Set values 73 amMsgDeleteElment 99 Error handling 73 amMsgDeleteElment 99 Distribution list interface functions 74 amMsgDeleteIopic 99 Get values 74 amMsgGetCSID 96 Get values 74 amMsgGetDatalength 99 Error handling 74 amMsgGetDatalength 99 Publisher interface functions 75 amMsgGetBlanentCSID 10 Get values 75 amMsgGetElmentCSID 10 Get values 75 amMsgGetElmentCount 10 Subscriber interface functions 76 amMsgGetFiller 10 Open and close 76 amMsgGetFiller 10 Subscriber interface functions 76 amMsgGetFiller 10 Open and close 76 amMsgGetFiller 10 Subscriber interface functions 76 amMsgGetFiller 10 Open and close 76 amMsgGetFiller 1	Receive and browse	. 73	amMsgAddTopic
Set values 73 amMsgDeleteElement Error handling 73 amMsgDeleteTilter Distribution list interface functions 74 amMsgDeleteTopic Send. 74 amMsgGetCoSID Get values 74 amMsgGetDataOffset Error handling 74 amMsgGetDataLength Publisher interface functions. 75 amMsgGetElement Open and close 75 amMsgGetElementCoSID 1 Publish 75 amMsgGetElementCosID 1 Get values 75 amMsgGetElementCosID 1 Subscriber interface functions 76 amMsgGetElementCount 1 Open and close 76 amMsgGetFilter 1 Set value 76 amMsgGetFilter 1 Get values.	Set values 73 amMsgDeleteFlement 99 Error handling 73 amMsgDeleteFlement 99 Distribution list interface functions 74 amMsgDeleteVamedElement 97 Open and close 74 amMsgGelCorrelld 98 Send. 74 amMsgGelDataLength 99 Get values 74 amMsgGelDataLength 99 Publisher interface functions 75 amMsgGelElement 99 Publish 75 amMsgGetElement 99 Publish 75 amMsgGetElementCount 100 Get values 75 amMsgGetElementCount 100 Subscriber interface functions 76 amMsgGetFliter 100 Subscriber interface functions 76 amMsgGetFliter 100 Subscriber interface functions 76 amMsgGetFlormat 100 Subscriber interface functions 76 amMsgGetTomat 100 Subscriber interface functions 76 amMsgGetNamedElement 100 Set value 76	Get values	. 73	amMsgClearErrorCodes 96
Error handling	Error handling	Set values	. 73	
Distribution list interface functions	Distribution list interface functions	Error handling	. 73	
Open and close	Open and close 74 amMsgGetCSID 98 Send. 74 amMsgGetCSID 98 Get values 74 amMsgGetDatal.ength 98 Error handling 74 amMsgGetBatal.ength 99 Publish interface functions 75 amMsgGetBlementCSID 19 Open and close 75 amMsgGetBlementCSID 10 Get values 75 amMsgGetBlementCSID 10 Error handling 75 amMsgGetBenentCSID 10 Subscriber interface functions 76 amMsgGetBenentCGID 10 Open and close 76 amMsgGetBenentCount 10 Open and close 76 amMsgGetBenentCount 10 Open and close 76 amMsgGetBerfilter 10 Open and close 76 amMsgGetBerford 10 Sevalue 76 amMsgGetBerford 10 Error handling 76 amMsgGetBerford 10 Set value 77 amMsgGetNamedElement 10 Erro			
Send.	Send. .74 amMsgGetCCSID .99 Get values .74 amMsgGetCorrelld .99 Error handling .74 amMsgGetDataLength .99 Publisher interface functions .75 amMsgGetElement .99 Open and close .75 amMsgGetElement CSID .10 Get values .75 amMsgGetElement Count .10 Error handling .75 amMsgGetElement Count .10 Subscriber interface functions .76 amMsgGetEliner .10 Copen and close .76 amMsgGetFlormat .10 Subscriber interface functions .76 amMsgGetFlormat .10 Get values .76 amMsgGetFlormat .10 Get values .76 amMsgGetBarelformat .10 Set value .76 amMsgGetBarelformat .10 Get values .77 amMsgGetNamedElement .10 Error handling .76 amMsgGetNamedElement .10 Error handling .77 amMsgGetNamedElement			amMsgDeleteTopic
Get values. 74 Publisher interface functions. 75 Open and close 75 Open and close 75 Open and close 75 Open and close 75 Get values. 75 Get values. 76 Get values. 76 Get values. 76 Get values. 76 Get values. 77 Get values. 77 Get values. 77 Get values. 77 Set value 76 Get values. 77 Get values. 77 Get values. 77 Get value 78 Get value 77 Get value 78 Get Value 78 Get Value 78 Get Value 79 Get Value 78 Get Value 79 Get	Get values 74 amMsgGetDataLength 99 Publisher interface functions 75 amMsgGetDataOffset 99 Open and close 75 amMsgGetEllement 99 Publish 75 amMsgGetEllementCCSID 100 Get values 75 amMsgGetEllementCount 100 Error handling 75 amMsgGetFliter 100 Subscriber interface functions 76 amMsgGetFliter 100 Open and close 76 amMsgGetFliterCount 101 Broker messages 76 amMsgGetFlorroupStatus 100 Get values 76 amMsgGetInsgGet 100 Error handling 76 amMsgGetNagGet 100 Error handling 76 amMsgGetNamedElement 100 Get values 77 amMsgGetNamedElement 100 Set value 77 amMsgGetNamedElement 100 High-level functions 78 amMsgGetNamedElement 100 Sest value 77 amMsgGetPopicCount 100	Send	. 74	amMsgGetCCSID
Error handling 74 amMsgGetDataLength. Publisher interface functions 75 amMsgGetEllement 1 Open and close 75 amMsgGetEllement 1 Get values 75 amMsgGetEllement 1 Error handling 75 amMsgGetEllementCount 1 Error handling 75 amMsgGetEllementCount 3 Open and close 76 amMsgGetFilter 0 1 Broker messages 76 amMsgGetFilter 0 1 Broker messages 76 amMsgGetForupStatus 3 Get value 76 amMsgGetLastError 3 Error handling 76 amMsgGetLastError 3 Error handling 76 amMsgGetLastError 3 Get value 77 amMsgGetNamedElement 3 Get value 77 amMsgGetNamedElement 3 Get value 77 amMsgGetNamedElement 3 Set value 77 amMsgGetNamedElement 3 Error handling 77 amMsgGetNamedElement 3 Set value 78 amMsgGetTopic 3 Error handling 79 amMsgGetTopic 4 High-level functions 82 amMsgGetTopic 4 Bession interface functions 82 amMsgGetTopic 0 1 Error handling 83 amMsgGetTopic 0 1 Error handling 84 amMsgGetTopic 0 1 Error handling 85 amMsgGetTopic 0 1 Error handling 86 amMsgGetTopic 0 1 Error handling 87 amMsgGetTopic 0 1 Error handling 88 amMsgSetClose 88 amMsgSetClose 1 Error handling 89 amMsgSetClose 1 Error handling 97 amMsgGetTopic 0 1 Error handling 1 1 Error handl	Error handling 74 amMsgGetDataLength 99 Voben and close 75 amMsgGetBateOffset 99 Open and close 75 amMsgGetElement 99 Publish 75 amMsgGetElement CSID 100 Get values 75 amMsgGetElementCount 100 Error handling 76 amMsgGetFilter Count 101 Open and close 76 amMsgGetFormat 100 Get values 76 amMsgGetFormat 100 Set value 76 amMsgGetLastError 100 Set value 76 amMsgGetNamgGetWangdd 100 Foror handling 76 amMsgGetNamedElement 100 For values 77 amMsgGetNamedElement 100 Get values 77 amMsgGetNamedElement 100 Error handling 77 amMsgGetNamedElement 100 Error handling 77 amMsgGetNamedElement 100 Error handling 77 amMsgGetNamedElementCont 100 <			amMsgGetCorrelId
Publisher interface functions. 75 Open and close 75 Open and close 75 Open and close 75 Open and close 75 Get values. 75 Get values. 75 Subscriber interface functions 76 Open and close 76 Get values. 76 Get values. 76 Get values. 76 Get value 76 Get value 77 Ge	Publisher interface functions			
Open and close 75 amMsgGetElement Publish 75 amMsgGetElementCOID Get values 75 amMsgGetElementCount Error handling 75 amMsgGetElementCount Open and close 76 amMsgGetFilter Open and close 76 amMsgGetFilterCount Broker messages 76 amMsgGetForuptStatus Get values 76 amMsgGetHorpat Set value 76 amMsgGetNagdd Dilcy interface functions 77 amMsgGetNamedElement Get values 77 amMsgGetNamedElementCount Brolevalue 77 amMsgGetNamedElementCount Get value 77 amMsgGetNamedElementCount Brolevalue 77 amMsgGetTopicCount Brolevalue 77 amMsgGetTopic Brolevalue 2 <t< td=""><td>Open and close .75 amMsgGetElement .99 Publish .75 amMsgGetElementCCSID 100 Get values .75 amMsgGetElementCount 100 Subscriber interface functions .76 amMsgGetFilter 100 Open and close .76 amMsgGetFilter 100 Broker messages .76 amMsgGetForupStatus 100 Get values .76 amMsgGetGroupStatus 100 Set value .76 amMsgGetHosptatus 100 Error handling .76 amMsgGetHosptatus 100 Policy interface functions .77 amMsgGetName 10 Get values .77 amMsgGetNamedElement 10 Error handling .77 amMsgGetRomedElementCount 10 Set value .77 amMsgGetRomedElementCoun</td><td></td><td></td><td>amMsgGetDataOffset</td></t<>	Open and close .75 amMsgGetElement .99 Publish .75 amMsgGetElementCCSID 100 Get values .75 amMsgGetElementCount 100 Subscriber interface functions .76 amMsgGetFilter 100 Open and close .76 amMsgGetFilter 100 Broker messages .76 amMsgGetForupStatus 100 Get values .76 amMsgGetGroupStatus 100 Set value .76 amMsgGetHosptatus 100 Error handling .76 amMsgGetHosptatus 100 Policy interface functions .77 amMsgGetName 10 Get values .77 amMsgGetNamedElement 10 Error handling .77 amMsgGetRomedElementCount 10 Set value .77 amMsgGetRomedElementCoun			amMsgGetDataOffset
Publish 75 amMsgGetElementCCsID Get values 75 amMsgGetElementCount Error handling 75 amMsgGetFilter Open and close 76 amMsgGetFilter Open and close 76 amMsgGetFilterCount Broker messages 76 amMsgGetForomat Get values 76 amMsgGetHastError Get value 76 amMsgGetHastError Error handling 76 amMsgGetNagId 1 Policy interface functions 77 amMsgGetName 1 Get values 77 amMsgGetNamedElement 1 Set value 77 amMsgGetNamedElement 1 Berror handling 77 amMsgGetNamedElementCount 1 Bigh-level functions 78 amMsgGetRoportCode 1 High-level functions 82 amMsgGetTopic Count 1 Session interface functions 82 amMsgGetTopic Count 1 amSesClearErrorCodes 82 amMsgGetCorcill 1 amSesClearErrorCodes	Publish .75 amMsgGetElementCCSID 100 Get values .75 amMsgGetElementCount 100 Error handling .75 amMsgGetFlementCount 100 Open and close .76 amMsgGetFlerCount 101 Broker messages .76 amMsgGetFormat 100 Get values .76 amMsgGetFormat 100 Set value .76 amMsgGetLastError 100 Error handling .76 amMsgGetNsgld 100 Policy interface functions .77 amMsgGetNamed 10 Get values .77 amMsgGetNamedElement 10 Set value .77 amMsgGetNamedElementCount 100 Error handling .77 amMsgGetTopic 10 High-level functions .78 amMsgGetTopic 10 High-level functions .82 amMsgGetTopic 10 samSesBegin .82 amMsgGetTopic 10 amSesBegin .82 amMsgGetTopic 10 samSesB			amMsgGetElement
Get values	Get values .75 amMsgGetEllementCount 100 Subscriber interface functions .76 amMsgGetFliter 100 Open and close .76 amMsgGetFliter 100 Broker messages .76 amMsgGetFoomat 100 Get values .76 amMsgGetFormat 100 Set value .76 amMsgGetLastError 100 Error handling .76 amMsgGetName 100 Policy interface functions .77 amMsgGetName 10 Get values .77 amMsgGetNamedElement 10 Set value .77 amMsgGetRoportCode 10 Error handling .77 amMsgGetRoportCode 10 High-level functions .78 amMsgGetTopic 10 Error handling .77 amMsgGetTopic 10 Chapter 5. C object interface reference 81 amMsgGetTopic 10 High-level functions .82 amMsgGetTopic 10 session interface functions .82 amMsgGetTopic			
Error handling	Error handling			amMsgGetElementCount
Subscriber interface functions 76 amMsgGetFilter 76 amMsgGetFilter 76 amMsgGetFilter 76 amMsgGetFormat 77 amMsgGetFormat 77 amMsgGetForupStatus 77 amMsgGetRoroupStatus 77 amMsgGetNameGlement 77 amMsgGetRoroupStatus 77 amMsgGetNameGlement 77 amMsgGetNameGlement 77 amMsgGetNameGlement 77 amMsgGetRoroupStatus 78 amMsgSetGupStatus 78 amSesCreateSubscriber 78 amSesCreateSubscriber 78 amSesDeleteNessage 78 amSesDeleteNessage 78 amSesDeleteNessage 79 amSesDeleteSubscribe 79 amSesDeleteSubscribe 79 amSesDeleteSubscribe 79 amSesDeleteSubscribe 79 amSesDeleteSubscribe 79 amSesDeletSubscribe 79	Subscriber interface functions 76 amMsgGetFilter 100 Open and close 76 amMsgGetFilterCount 101 Broker messages. 76 amMsgGetFormat 100 Get values. 76 amMsgGetGroupStatus 100 Set value 76 amMsgGetMsgId 100 Error handling 76 amMsgGetNamedElement 100 Get values. 77 amMsgGetNamedElement 100 Set value 77 amMsgGetNamedElementCount 100 Error handling 77 amMsgGetTopic 100 High-level functions 78 amMsgGetTopic 100 Error handling 77 amMsgGetTopic 100 Chapter 5. C object interface reference 81 amMsgGetTopic 100 Chapter 5. C object interface reference 81 amMsgGetTopic 100 Session interface functions 82 amMsgGetTopic 100 amSesBegin 82 amMsgRestTopic 100 amSesClose 83 amMsgRestDetCorl			amMsgGetEncoding
Open and close Broker messages. 76 Broker messages. 76 Get values. 76 Set value 76 Set value 76 Error handling 76 Error handling 77 Get values. 78 Get value 79 Get values. 79 Get values. 70 Get values. 70 Get values. 71 Get values. 72 Get values. 73 Get value 74 Get values. 75 Get value 75 Get value 76 Get values. 77 Get values. 78 Get value 79 Get values. 79 Get values. 70 Get values. 70 Get values. 71 AmMsgGetNamedElement 71 AmMsgGetRoporCode 71 AmMsgGetRoporCode 72 MmsgGetTopic 73 MmsgGetTopic 74 MmsgGetTopic 75 MmsgGetTopic 75 MmsgGetTopic 75 MmsgGetTopic 75 MmsgGetTopic 75 MmsgGetTopic 75 MmsgGetTopic 76 MmsgGetTopic 76 MmsgGetTopic 77 MmsgGetTopic 76 MmsgGetTopic 77 MmsgGetTopic 78 MmsgGetTopic 78 MmsgGetTopic 79 MmsgGetTopic 70 MmsgGetT	Open and close 76 amMsgGetFirerCount 101 Broker messages. 76 amMsgGetFormat. 102 Get values. 76 amMsgGetGetGroupStatus 102 Set value 76 amMsgGetMsgld 103 Error handling 76 amMsgGetNamedElement 106 Get values. 77 amMsgGetNamedElementCount 105 Set value 77 amMsgGetRoportCode 106 Error handling 77 amMsgGetTopic 10 High-level functions 78 amMsgGetTopic 10 High-level functions 81 amMsgGetTopic 10 AmmsegSetDate functions 82 amMsgGetTopic 10 amsesCleaterrorCodes 81 amMsgGetTopicCount 10 session interface functions 82 amMsgReadBytes 10 amSesClareTrorCodes 82 amMsgSetCortplic 10 amSesClotaer functions 82 amMsgSetCortplid 10 amSesClotaer functions 83 amMsgSetCortplid 1			
Broker messages. 76 amMsgGetFormat 1 Get values 76 amMsgGetLastError 1 Error handling 76 amMsgGetLastError 1 Error handling 76 amMsgGetName 1 Oet values 77 amMsgGetName 1 Get values 77 amMsgGetName 1 Set value 77 amMsgGetNamedElement 1 Set value 77 amMsgGetNamedElement 1 Error handling 77 amMsgGetNamedElement 1 Error handling 77 amMsgGetReportCode 1 High-level functions 78 amMsgGetPopic 2 amMsgGetTopic 3 amMsgReadBytes 1 amMsgGetTopic 4 amMsgGetTopic 5 amMsgReadBytes 1 amMsgGetTopic 5 amMsgReadBytes 1 amMsgGetTopic 6 amMsgGetTopic 7 amMsgGetTopic 7 amMsgGetTopic 7 amMsgGetTopic 8 amMsgReadBytes 1 amMsgSetCCSID 1 amMsgSetCCSID 1 amMsgSetCCSID 1 amMsgSetCorelld 1 amSesCreate 83 amMsgSetDataOffset 1 amSesCreate 84 amMsgSetEncoding 1 amSesCreatePolict 84 amMsgSetEncoding 1 amSesCreatePolicty 84 amMsgSetGroupStatus 1 amSesCreatePolicy 84 amMsgSetGroupStatus 1 amSesCreateReceiver 85 amMsgSetTopic 8 amSesCreateReceiver 86 amMsgSetTopic 8 amSesCreateSubscriber 86 amMsgSetTopic 8 amSesDeleteNessage 87 amMsgGetPublTimeStamp 8 amSesDeleteNessage 88 amSndClearErrorCodes 8 amSesDeleteReceiver 88 amSndClearErrorCodes 8 amSndCetLooding 8 amSndGetLastError	Broker messages. 76 amMsgGetFormat 102 Get values. 76 amMsgGetLastError 102 Set value 76 amMsgGetLastError 102 Error handling 76 amMsgGetMsgld 102 Get values. 77 amMsgGetNamedElement 102 Get value 77 amMsgGetNamedElementCount 102 Error handling 77 amMsgGetTopic 100 High-level functions 78 amMsgGetTopic 100 High-level functions 81 amMsgGetTopicCount 107 Session interface functions 82 amMsgGetTopicCount 100 amSescion interface functions 82 amMsgReadBytes 100 amSesClearErrorCodes 82 amMsgReadBytes 100 amSesClearErrorCodes 82 amMsgSetCorrelld 10 amSesClose 83 amMsgSetCorrelld 10 amSesCommit 83 amMsgSetDatoOffset 10 amSesCreateDistList 84 amMsgSetElncoding 105 <td></td> <td></td> <td></td>			
Get value 76 amMsgGetCroupStatus Set value 76 amMsgGetMsgGetLastError 1 Error handling 76 amMsgGetMsgId 1 Policy interface functions 77 amMsgGetName 1 Get values 77 amMsgGetNamedElement Count 1 Set value 77 amMsgGetRoportCode 1 High-level functions 78 amMsgGetTopic 1 High-level functions 81 amMsgGetTopic 1 Session interface functions 82 amMsgGetTopicCount 1 Session interface functions 82 amMsgGetTopicCount 1 Session interface functions 82 amMsgGetTopicCount 1 amSesCleair Error Godes 82 amMsgGetTopicCount 1 amSesClear Error Godes 82 amMsgReadBytes 1 amSesClear Error Godes 82 amMsgSetCrorelld 1 amSesCreate Elustitist 83 amMsgSetElementCSID 1 amSesCreate Elustistist 84 amMsgSetElocoding	Get values 76 amMsgGetGroupStatus 100 Set value 76 amMsgGetLastError 100 Error handling 76 amMsgGetName 100 Policy interface functions 77 amMsgGetNamedElement 100 Get values 77 amMsgGetNamedElementCount 100 Set value 77 amMsgGetRoportCode 100 High-level functions 78 amMsgGetTopic 100 High-level functions 81 amMsgGetTopic 100 Session interface functions 82 amMsgGetTopicCount 100 Session interface functions 82 amMsgReadBytes 100 amSesClearErrorCodes 82 amMsgReadBytes 100 amSesClearErrorCodes 82 amMsgSetCOSID 10 amSesClose 83 amMsgSetCosID 10 amSesCommit 83 amMsgSetCorrell 10 amSesCreate 83 amMsgSetCorrell 10 amSesCreateMessage 84 amMsgSetTopotCode 111			
Set value 76 amMsgGetLastError Error handling 76 amMsgGetMsgld 1 Policy interface functions 77 amMsgGetNamed 1 Get values 77 amMsgGetNamedElement 1 Set value 77 amMsgGetReportCode 1 Error handling 77 amMsgGetReportCode 1 High-level functions 78 amMsgGetTopic 1 Bession interface reference 81 amMsgGetTopic 1 Session interface functions 82 amMsgGetTopic 1 session interface functions 82 amMsgGetTopicCount 1 amSesBegin 82 amMsgReadBytes 1 amSesBegin 82 amMsgReadBytes 1 amSesClearErrorCodes 82 amMsgSetCCSID 1 amSesClearErrorCodes 82 amMsgSetCCSID 1 amSesClose 83 amMsgSetDataOffset 1 amSesCreateDistList 84 amMsgSetEncoding 1 amSesCreatePublisher </td <td>Set value 76 amMsgGetLastError 105 Error handling 76 amMsgGetNsgId 105 Policy interface functions 77 amMsgGetName 106 Get values 77 amMsgGetNamedElement 106 Set value 77 amMsgGetNamedElementCount 106 Error handling 77 amMsgGetTopic 106 High-level functions 78 amMsgGetTopic 106 High-level functions 81 amMsgGetTopic (Ount 100 Session interface functions 82 amMsgGetTopic Count 100 Session interface functions 82 amMsgGetTopic Count 100 amSesBegin 82 amMsgReadBytes 100 amSesGetTeate Count 100 amMsgSetGetTopic Count 100 amSesGetBegin 82 amMsgReadBytes 100 amSesGetBegin 82 amMsgReetDout 100 amSesCleatrerorOcdes 83 amMsgSetCSID 100 amSesComait 83 amMsgSetElementCCSID <td< td=""><td>Get values</td><td>76</td><td>amMsgGetGroupStatus 102</td></td<></td>	Set value 76 amMsgGetLastError 105 Error handling 76 amMsgGetNsgId 105 Policy interface functions 77 amMsgGetName 106 Get values 77 amMsgGetNamedElement 106 Set value 77 amMsgGetNamedElementCount 106 Error handling 77 amMsgGetTopic 106 High-level functions 78 amMsgGetTopic 106 High-level functions 81 amMsgGetTopic (Ount 100 Session interface functions 82 amMsgGetTopic Count 100 Session interface functions 82 amMsgGetTopic Count 100 amSesBegin 82 amMsgReadBytes 100 amSesGetTeate Count 100 amMsgSetGetTopic Count 100 amSesGetBegin 82 amMsgReadBytes 100 amSesGetBegin 82 amMsgReetDout 100 amSesCleatrerorOcdes 83 amMsgSetCSID 100 amSesComait 83 amMsgSetElementCCSID <td< td=""><td>Get values</td><td>76</td><td>amMsgGetGroupStatus 102</td></td<>	Get values	76	amMsgGetGroupStatus 102
Error handling Policy interface functions Get values. 77 amMsgGetNamed Element Set value 77 amMsgGetNamed Element 78 amMsgGetNamed Element 79 amMsgGetNamed Element 79 amMsgGetRamed Element 70 amMsgGetRamed Element 71 amMsgGetRoportCode 72 amMsgGetTopic 73 amMsgGetTopic 74 amMsgGetTopic 75 amMsgGetTopic 76 amMsgGetTopic 76 amMsgGetTopic 77 amMsgGetTopic 78 amMsgGetTopic 79 amMsgGetTopic 70 amMsgGetTopic 70 amMsgGetTopic 70 amMsgGetTopic 71 amMsgGetTopic 71 amMsgGetTopic 72 amMsgGetTopic 73 amMsgGetTopic 74 amMsgGetTopic 75 amMsgGetTopic 76 amMsgReadBytes 76 amMsgSetCrorulld 76 amMsgSetCrorulld 76 amMsgSetCrorulld 77 amMsgReadBytes 77 amMsgReadBytes 70 amMsgReadBytes 70 amMsgReadBytes 70 amMsgSetCrount 71 amMsgGetTopic 71 amMsgSetCrount 71 amMsgCetCrount 71 amMsgCetCr	Error handling			amMsgGetTastFrror
Policy interface functions	Policy interface functions			amMcaCatMcaId 103
Get values	Get values 77 amMsgGetNamedElement 100 Set value 77 amMsgGetRamedElementCount 105 Error handling 77 amMsgGetReportCode 105 High-level functions 78 amMsgGetTopic 100 Chapter 5. C object interface reference 81 amMsgGetTopic 100 Session interface functions 82 amMsgGetTopicCount 107 amSesBegin 82 amMsgReadBytes 107 amSesClearErrorCodes 82 amMsgSetCCSID 105 amSesClose 83 amMsgSetCorrelld 108 amSesCommit 83 amMsgSetElementCCSID 108 amSesCreate 83 amMsgSetElementCCSID 105 amSesCreateMessage 84 amMsgSetElementCCSID 105 amSesCreatePolistList 84 amMsgSetTornat 110 amSesCreatePolistLisher 85 amMsgSetTornat 111 amSesCreatePolicy 84 amMsgSetPortCode 111 amSesCreateSubscriber 86 amMsgSetPo			
Set value	Set value 77 amMsgGetRamedElementCount 105 Error handling 77 amMsgGetReportCode 106 High-level functions 78 amMsgGetTopic 106 amMsgGetType 106 amMsgGetType 106 Chapter 5. C object interface reference 81 amMsgGetTopic Count 107 Session interface functions 82 amMsgReadBytes 107 amSesBegin 82 amMsgReadBytes 107 amSesClearErrorCodes 82 amMsgReatBytes 107 amSesClose 83 amMsgSetCCSID 108 amSesClose 83 amMsgSetCorrelld 108 amSesCreate 83 amMsgSetElementCCSID 108 amSesCreatePistList 84 amMsgSetElementCCSID 109 amSesCreatePistList 84 amMsgSetEncoding 10 amSesCreatePublisher 85 amMsgSetGroupStatus 110 amSesCreateSubscriber 86 amMsgSetType 111 amSesCreateSubscriber 86 mMsgSetType			
Error handling 77 amMsgGetReportCode 19 digh-level functions 78 amMsgGetTopic 19 amMsgGetTopic 19 amMsgGetTopic 20 amMsgGetTopic 20 amMsgGetTopic 20 amMsgGetTopic 20 amMsgGetTopic 20 amMsgReadBytes 20 amMsgReadBytes 20 amMsgReadBytes 20 amMsgReset 20 amMsgReset 20 amMsgReset 20 amMsgReset 20 amMsgReset 20 amMsgSetCorrelld 20 amSesClose 20 amMsgSetCorrelld 20 amSesCommit 20 amMsgSetDataOffset 20 amMsgSetDataOffset 20 amMsgSetDataOffset 20 amMsgSetDataOffset 20 amMsgSetDataOffset 20 amMsgSetElementCCSID 20 amSesCreate 20 amMsgSetElementCCSID 20 amMsgSetTopic 20 amMsgSetElementCCSID 20 amMsgSetTopic 20 amMsgSetFormat 20 amMsgSetFormat 20 amMsgSetFormat 20 amMsgSetFormat 20 amMsgSetFormat 20 amMsgSetFormat 20 amMsgSetForpotCode 20 amMsgSetTopic 20	Error handling 77 amMsgGetReportCode 105 High-level functions 78 amMsgGetTopic 106 Chapter 5. C object interface reference 81 amMsgGetTopic Count 107 Session interface functions 82 amMsgReadBytes 107 amSesBegin 82 amMsgSetCoSID 108 amSesClose 83 amMsgSetCoSID 108 amSesClose 83 amMsgSetDataOffset 108 amSesCreate 83 amMsgSetElementCCSID 108 amSesCreateDistList 84 amMsgSetElementCCSID 109 amSesCreatePostelpistList 84 amMsgSetFormat 110 amSesCreatePolicy 84 amMsgSetForupStatus 111 amSesCreatePublisher 85 amMsgSetToptCode 111 amSesCreateSubscriber 86 amMsgWriteBytes 112 amSesCreateSubscriber 86 amMsgRetReportCode 111 amSesDeleteDistList 87 AmMsgAddStramName 112 amSesDeletePolicy 88 <t< td=""><td></td><td></td><td></td></t<>			
High-level functions 78 amMsgGetTopic amMsgGetType. 1 Chapter 5. C object interface reference 81 amMsgGetTopicCount amMsgReadBytes amSesBegin 82 amMsgReadBytes amSesBegin 82 amMsgReadBytes amSesClearErrorCodes 82 amMsgSetCCSID 1 amSesClose 83 amMsgSetCorrelId 1 amSesCreate 83 amMsgSetDataOffset 1 amSesCreate 83 amMsgSetElamentCCSID 1 amSesCreate 84 amMsgSetEncoding 1 amSesCreatePolistList 84 amMsgSetEncoding 1 amSesCreatePolicy 84 amMsgSetFormat 1 amSesCreatePublisher 85 amMsgSetRoportCode 1 amSesCreateSubscriber 86 amSesCreateSubscriber 86 amSesCreateSubscriber 86 amSesCreateSubscriber 87 amSesDelete 887 amMsgSetTreamName 888 amSesDeletePolicy 88 amSesDeleteReceiver 88 amSndClearErrorCodes 88 amSndClearErrorCodes 89 amSndGetEncoding 89 amSndGetEncoding 89 amSndGetEnastError	High-level functions 78 amMsgGetTopic amMsgGetTopic amMsgGetTopic amMsgGetTopic amMsgGetTopicCount 106 Chapter 5. C object interface reference 81 amMsgGetTopicCount 107 Session interface functions 82 amMsgReadBytes 107 amSesBegin 82 amMsgReadBytes 107 amSesClearErrorCodes 82 amMsgSetCCSID 108 amSesClose 83 amMsgSetDataOffset 108 amSesCommit 83 amMsgSetDataOffset 108 amSesCreate 83 amMsgSetEmatCSID 108 amSesCreateDistList 84 amMsgSetEncoding 109 amSesCreateDistList 84 amMsgSetEncoding 10 amSesCreatePublisher 85 amMsgSetGroupStatus 111 amSesCreatePublisher 85 amMsgSetReportCode 111 amSesCreateSender 86 amMsgSetType 111 amSesDelete 86 mssage interface helper macros 112 amSesDeleteDistList 87 AmMsgAddStreamName 113 <		. //	
Chapter 5. C object interface reference 81 amMsgGetTopicCount amMsgGetTopicCount amMsgReson interface functions 82 amMsgReson amMsgReson amMsgReson amMsgReson amMsgReson amMsgSetCoSID amSesClearErrorCodes 82 amMsgSetCorrelId. 3 amSesClose 83 amMsgSetDataOffset 3 amMsgSetDataOffset 3 amMsgSetDataOffset 4 amMsgSetDataOffset 4 amMsgSetDataOffset 5 amSesCreate 83 amMsgSetEncoding 3 amMsgSetEncoding 4 amMsgSetFormat 5 amMsgSetFormat 6 amMsgSetFormat 7 amSesCreatePolicy 84 amMsgSetFormat 8 amMsgSetFormat 8 amMsgSetGroupStatus 8 amMsgSetGroupStatus 8 amMsgSetReportCode 8 amMsgSetReportCode 8 amMsgSetType 8 amMsgSetType 8 amMsgSetType 8 amMsgSetPopt 8 amMsgSetType 8 amMsgSetPopt 8 amMsgSetPopt 8 amMsgSetPopt 8 amMsgSetPopt 8 amMsgSetPubTimeStamp 8 amSesDelete 9 states 8 amSndClose 8 amSesDeletePublisher 8 amSndClose 8 amSndClose 8 amSndClose 8 amSndClose 8 amSndCetCSID 8 amSesDeleteSender 8 amSndGetCSID 8 amSndGetCSID 8 amSndGetLastError 8	Chapter 5. C object interface reference 81 amMsgGetToppicCount 106 Session interface functions 82 amMsgReadBytes 107 amSesBegin 82 amMsgReset 107 amSesClearErrorCodes 82 amMsgSetCCSID 108 amSesClose 83 amMsgSetCorrelld 108 amSesCommit 83 amMsgSetEalcoffset 108 amSesCreate 83 amMsgSetElmcodfise 108 amSesCreateDistList 84 amMsgSetEncoding 109 amSesCreateMessage 84 amMsgSetFormat 110 amSesCreatePolicy 84 amMsgSetTopen 111 amSesCreatePolicy 84 amMsgSetType 111 amSesCreateReceiver 85 amMsgSetType 111 amSesCreateSubscriber 86 amMsgSetType 111 amSesDelete 87 AmMsgAddStreamName 112 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeletePublisher 88 amSndClearErrorCodes			
Chapter 5. C object interface reference81amMsgGetTopicCountSession interface functions82amMsgReadBytesamSesBegin82amMsgResetamSesClearErrorCodes82amMsgSetCSIDamSesClose83amMsgSetCorrelldamSesCommit83amMsgSetDataOffsetamSesCreate83amMsgSetElementCCSIDamSesCreateOistList84amMsgSetEncodingamSesCreateMessage84amMsgSetFormatamSesCreatePublisher85amMsgSetRoupStatusamSesCreatePublisher85amMsgSetTypeamSesCreateSender86amMsgSetTypeamSesCreateSubscriber86amMsgWriteBytesamSesDelete87AmMsgAddStreamNameamSesDeleteDistList87AmMsgGetPubTimeStampamSesDeletePolicy88sender interface functionsamSesDeletePublisher88amSndClearErrorCodesamSesDeleteReceiver88amSndClearErrorCodesamSesDeleteReceiver88amSndCloseamSesDeleteSender89amSndGetCCSIDamSesDeleteSubscriber89amSndGetCastError	Chapter 5. C object interface reference 81 amMsgGetTopicCount 107 Session interface functions 82 amMsgReadBytes 107 amSesBegin 82 amMsgReatBytes 107 amSesClose 83 amMsgSetCCSID 108 amSesClose 83 amMsgSetCorrelld 108 amSesCommit 83 amMsgSetDataOffset 108 amSesCreate 83 amMsgSetElementCCSID 108 amSesCreateDistList 84 amMsgSetFornat 109 amSesCreateMessage 84 amMsgSetFornat 110 amSesCreatePolicy 84 amMsgSetGroupStatus 110 amSesCreatePolicy 84 amMsgSetGroupStatus 111 amSesCreateReceiver 85 amMsgSetTope 111 amSesCreateReceiver 85 amMsgSetTope 111 amSesCreateSubscriber 86 amMsgWriteBytes 112 amSesDelete 87 AmMsgAddStreamName 112 amSesDeleteDistList 87 AmMsgGetPubTimeStamp	Error handling	. 77	amMsgGetReportCode
Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesClose amSesCommit amSesCreate amSesCreate amSesCreate amSesCreateDistList amSesCreatePublisher amSesCreateSubscriber amSesDeletePublisher amSesDeleteReceiver amSesDeletePolicy amSesDeleteReceiver amSesDeleteRec	Session interface functions 82 amMsgReadBytes 107 amSesBegin 82 amMsgReset 107 amSesClearErrorCodes 82 amMsgSetCCSID 108 amSesClose 83 amMsgSetCorrelld 108 amSesCommit 83 amMsgSetDataOffset 108 amSesCreate 83 amMsgSetElementCCSID 109 amSesCreateDistList 84 amMsgSetElementCCSID 109 amSesCreatePolistList 84 amMsgSetElementCCSID 109 amSesCreatePolistList 84 amMsgSetFormat 110 amSesCreatePolicity 84 amMsgSetFormat 111 amSesCreatePublisher 85 amMsgSetToupStatus 111 amSesCreateReceiver 85 amMsgSetType 111 amSesCreateSubscriber 86 amMsgWriteBytes 112 amSesDelete 87 AmMsgAddStreamName 113 amSesDeleteDistList 87 AmMsgGetStreamName 114 amSesDeletePolicy 88 amSndClearErrorCodes	Error handling	. 77	amMsgGetReportCode
amSesBegin 82 amMsgReset. amSesClearErrorCodes 82 amMsgSetCCSID amSesClose 83 amMsgSetCorrelId. amSesCommit 83 amMsgSetDataOffset 93 amMsgSetElementCCSID 94 amSesCreate 95 amMsgSetElementCCSID 95 amSesCreatePolicy 95 amSesCreatePolicy 96 amSesCreatePolicy 96 amSesCreatePolicy 97 amSesCreateSender 97 amSesCreateSender 98 amMsgSetTorupStatus 97 amSesCreateSender 98 amMsgSetTorupStatus 98 amMsgSetType 98 amSesCreateSender 98 amMsgSetType 98 amSesCreateSender 98 amMsgSetType 98 amSesCreateSender 98 amMsgSetType 98 amSesCreateSender 98 amMsgSetType 98 amSesDeleteDistList 97 amMsgGetPubTimeStamp 98 amSesDeleteDistList 97 amMsgGetPubTimeStamp 98 amSesDeletePolicy 98 amSesDeleteMessage 97 AmMsgGetStreamName 98 amSesDeleteReceiver 98 amSndClearErrorCodes 98 amSndClose 98 amSndClose 98 amSndClose 98 amSndClose 98 amSndGetCCSID 98 amSesDeleteSubscriber 99 amSndGetLastError	amSesBegin 82 amMsgReset. 107 amSesClearErrorCodes 82 amMsgSetCorEID 108 amSesClose 83 amMsgSetCorrelId. 108 amSesCommit 83 amMsgSetDataOffset 108 amSesCreate 83 amMsgSetElementCCSID 109 amSesCreateDistList 84 amMsgSetElementCCSID 109 amSesCreateMessage 84 amMsgSetFormat 110 amSesCreateMessage 84 amMsgSetFormat 111 amSesCreatePublisher 85 amMsgSetReportCode 111 amSesCreateReceiver 85 amMsgSetReportCode 111 amSesCreateReceiver 85 amMsgSetType 111 amSesCreateSubscriber 86 amMsgWriteBytes 112 amSesCreateSubscriber 86 Message interface helper macros 113 amSesDelete 87 AmMsgAddStreamName 113 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeletePolicy 88 amSndClearErrorCodes<	Error handling	. 77 . 78	amMsgGetReportCode
amSesClearErrorCodes amSesClose amSesClose amSesCommit amSesCommit amSesCreate amSesCreate amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDeleteDistList amSesDeletePublisher amSesDeleteReceiver amSesDeletePublisher amSesDeleteReceiver amSesDeletePublisher amSesDeleteReceiver amSesDeleteNessage amSesDeleteNessage amSesDeleteNessage amSesDeleteNessage amSesDeleteNessage amSesDeleteNessage amSesDeleteNessage amSesDeletePolicy amSesDeleteReceiver amSesDeleteSubscriber amSesDeletastError	amSesClearErrorCodes 82 amMsgSetCCSID 108 amSesClose 83 amMsgSetCorrelld. 108 amSesCommit 83 amMsgSetDataOffset 108 amSesCreate 83 amMsgSetEllementCCSID 109 amSesCreateDistList 84 amMsgSetEllementCCSID 109 amSesCreateMessage 84 amMsgSetEllementCCSID 109 amSesCreateMessage 84 amMsgSetFormat 110 amSesCreatePolicy 84 amMsgSetGroupStatus 110 amSesCreatePublisher 85 amMsgSetReportCode 111 amSesCreateReceiver 85 amMsgSetType 111 amSesCreateSender 86 amMsgWriteBytes 112 amSesCreateSubscriber 86 Message interface helper macros 113 amSesDelete 87 AmMsgAddStreamName 113 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeletePolicy 88 AmMsgGetStreamName 114 amSesDeletePolicy 88 amSn	Error handling	. 77 . 78	amMsgGetReportCode
amSesClose amSesCommit amSesCreate amSesCreate amSesCreate amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDeletePublisher amSesDeletePublisher amSesDeletePublisher amSesDeletePublisher amSesDeleteReceiver amSesDeletePublisher amSesDeleteReceiver amSesDeletePublisher amSesDeletePublisher amSesDeleteReceiver amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle	amSesClose 83 amMsgSetCorrelld. 108 amSesCommit 83 amMsgSetDataOffset 108 amSesCreate 83 amMsgSetElementCCSID 109 amSesCreateDistList 84 amMsgSetEncoding 109 amSesCreateMessage 84 amMsgSetFormat 110 amSesCreatePolicy 84 amMsgSetGroupStatus 111 amSesCreatePublisher 85 amMsgSetReportCode 111 amSesCreateReceiver 85 amMsgSetType 111 amSesCreateReceiver 85 amMsgWriteBytes 112 amSesCreateSubscriber 86 amMsgWriteBytes 112 amSesDelete 87 AmMsgAddStreamName 113 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeleteMessage 87 AmMsgGetStreamName 114 amSesDeletePolicy 88 amSndClearErrorCodes 115 amSesDeleteReceiver 88 amSndClearErrorCodes 115 amSesDeleteSender 89 amSndGetCoding	Error handling	. 77 . 78 81 . 82	amMsgGetReportCode
amSesCommit 83 amMsgSetDataOffset 1 amSesCreate 83 amMsgSetElementCCSID 1 amSesCreateDistList 84 amMsgSetEncoding 1 amSesCreateMessage 84 amMsgSetFormat 1 amSesCreatePolicy 84 amMsgSetGroupStatus 1 amSesCreatePublisher 85 amMsgSetType 1 amSesCreateSender 85 amMsgSetType 1 amSesCreateSender 86 amMsgSetType 1 amSesCreateSubscriber 86 Message interface helper macros 1 amSesDelete 87 AmMsgAddStreamName 1 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 1 amSesDeleteMessage 87 AmMsgGetStreamName 1 amSesDeletePolicy 88 Sender interface functions 1 amSesDeletePublisher 88 amSndClearErrorCodes 1 amSesDeleteReceiver 88 amSndClose 1 amSesDeleteSender 89 amSndGetCCSID 1 amSesDeleteSubscriber 89 amSndGetEncoding 1 amSesGetDistListHandle 89 amSndGetLastError	amSesCommit 83 amMsgSetDataOffset 108 amSesCreate 83 amMsgSetElementCCSID 105 amSesCreateDistList 84 amMsgSetElementCCSID 105 amSesCreateMessage 84 amMsgSetForoding 105 amSesCreatePolicy 84 amMsgSetGroupStatus 116 amSesCreatePublisher 85 amMsgSetReportCode 111 amSesCreateReceiver 85 amMsgSetType 111 amSesCreateSender 86 amMsgSetType 111 amSesCreateSender 86 Message interface helper macros 112 amSesDelete 87 AmMsgAddStreamName 113 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeleteMessage 87 AmMsgGetStreamName 114 amSesDeletePolicy 88 amSndClearErrorCodes 115 amSesDeletePublisher 88 amSndClearErrorCodes 115 amSesDeleteReceiver 88 amSndClearErrorCodes 115 amSesDeleteSender 89	Error handling	. 77 . 78 81 . 82 . 82	amMsgGetReportCode . 105 amMsgGetTopic . 106 amMsgGetType . 107 amMsgGetTopicCount . 107 amMsgReadBytes . 107 amMsgReset . 107
amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateReceiver amSesCreateSubscriber amSesDeleteDistList amSesDeletePolicy amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeletePublisher amSesDeleteReceiver amSesDeleteSender amSesDeleteSender amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteReceiver amSesDelete	amSesCreate 83 amMsgSetElementCCSID 105 amSesCreateDistList 84 amMsgSetEncoding 105 amSesCreateMessage 84 amMsgSetFormat 110 amSesCreatePolicy 84 amMsgSetGroupStatus 110 amSesCreatePublisher 85 amMsgSetReportCode 111 amSesCreateReceiver 85 amMsgSetType 111 amSesCreateSender 86 amMsgWriteBytes 112 amSesCreateSubscriber 86 Message interface helper macros 113 amSesDelete 87 AmMsgAddStreamName 113 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeleteMessage 87 AmMsgGetPubTimeStamp 113 amSesDeletePublisher 88 Sender interface functions 114 amSesDeletePublisher 88 amSndClearErrorCodes 115 amSesDeleteReceiver 88 amSndClose 115 amSesDeleteSender 89 amSndGetCCSID 116 amSesDeleteSubscriber 89 amSndGetEncoding 116 amSesGetDistListHandle 89 amSndGetLastError 116 amSesGetLastError 90 amSndGetName 117 amSesGetMessageHandle 90 amSndOpen 117	Error handling	. 77 . 78 81 . 82 . 82 . 82	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108
amSesCreateDistList 84 amMsgSetEncoding amSesCreateMessage 84 amMsgSetFormat amSesCreatePolicy 84 amMsgSetGroupStatus amSesCreatePublisher 85 amMsgSetReportCode amSesCreateReceiver 85 amMsgSetType amSesCreateSender 86 amMsgWriteBytes amSesCreateSubscriber 86 Message interface helper macros amSesDelete 87 AmMsgAddStreamName amSesDeleteDistList 87 AmMsgGetPubTimeStamp amSesDeleteMessage 87 AmMsgGetStreamName amSesDeletePolicy 88 Sender interface functions amSesDeletePublisher 88 amSndClearErrorCodes amSesDeleteReceiver 88 amSndClose amSesDeleteSender 89 amSndGetCCSID amSesDeleteSubscriber 89 amSndGetEncoding amSesGetDistListHandle 89 amSndGetLastError	amSesCreateDistList amSesCreateMessage 84 amMsgSetFormat 110 amSesCreatePolicy 84 amMsgSetGroupStatus 1110 amSesCreatePublisher 85 amMsgSetReportCode 1111 amSesCreateReceiver 85 amMsgSetType 1112 amSesCreateSender 86 amMsgWriteBytes 1112 amSesCreateSubscriber 86 Message interface helper macros 1113 amSesDelete 1114 amSesDeleteDistList 1115 amSesDeleteMessage 1116 amSesDeleteMessage 1117 amSesDeletePolicy 1118 amSesDeletePolicy 1119 amSesDeleteReceiver 1119 amSesDeleteReceiver 1119 amSesDeleteSender 1119 amSesDeleteSubscriber 1110 amSesDeleteSubscriber 1111 amSesDeleteSubscriber 1112 amSesDeleteSubscriber 1115 amSesDeleteSubscriber 1116 amSesDeleteSubscriber 1117 amSesGetDistListHandle 1117 amSesGetLastError 1116 amSesGetLastError 1116 amSesGetMessageHandle 1117 amSesGetMessageHandle 1117 amSesGetMessageHandle	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108 amMsgSetCorrelId 108
amSesCreateMessage 84 amMsgSetFormat amSesCreatePolicy 84 amMsgSetGroupStatus amSesCreatePublisher 85 amMsgSetReportCode amSesCreateReceiver 85 amMsgSetType amSesCreateSender 86 amMsgWriteBytes 86 amMsgWriteBytes 87 amSesCreateSubscriber 86 Message interface helper macros 87 amSesDelete 87 amMsgAddStreamName 88 amSesDeleteDistList 87 amMsgGetPubTimeStamp 88 AmMsgGetPubTimeStamp 88 Sender interface functions 88 amSesDeletePolicy 88 Sender interface functions 88 amSesDeleteReceiver 88 amSndClearErrorCodes 88 amSndClose 89 amSndGetCCSID amSesDeleteSubscriber 89 amSndGetEncoding amSesDeleteNate 89 amSndGetEncoding amSesDeleteNate 89 amSndGetLastError	amSesCreateMessage 84 amMsgSetFormat 110 amSesCreatePolicy 84 amMsgSetGroupStatus 1110 amSesCreatePublisher 85 amMsgSetReportCode 1111 amSesCreateReceiver 85 amMsgSetType 1112 amSesCreateSender 86 amMsgWriteBytes 1112 amSesCreateSubscriber 86 Message interface helper macros 1113 amSesDelete 87 AmMsgAddStreamName 1113 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 1113 amSesDeleteMessage 87 AmMsgGetStreamName 1114 amSesDeletePolicy 88 Sender interface functions 1115 amSesDeletePublisher 88 amSndClearErrorCodes 1115 amSesDeleteReceiver 88 amSndClose 1115 amSesDeleteSender 89 amSndGetCCSID 1116 amSesDeleteSubscriber 89 amSndGetEncoding 1116 amSesGetDistListHandle 89 amSndGetLastError 1116 amSesGetLastError 90 amSndGetName 1117 amSesGetMessageHandle 90 amSndOpen 1117	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108 amMsgSetCorrelId 108 amMsgSetDataOffset 108
amSesCreatePolicy 84 amMsgSetGroupStatus amSesCreatePublisher 85 amMsgSetReportCode amSesCreateReceiver 85 amMsgSetType amSesCreateSender 86 amMsgWriteBytes amSesCreateSubscriber 86 Message interface helper macros amSesDelete 87 AmMsgAddStreamName amSesDeleteDistList 87 AmMsgGetPubTimeStamp amSesDeleteMessage 87 AmMsgGetStreamName amSesDeletePolicy 88 Sender interface functions amSesDeletePublisher 88 amSndClearErrorCodes amSesDeleteReceiver 88 amSndClose amSesDeleteSender 89 amSndGetCCSID amSesDeleteSubscriber 89 amSndGetEncoding amSesGetDistListHandle 89 amSndGetLastError	amSesCreatePolicy 84 amMsgSetGroupStatus 110 amSesCreatePublisher 85 amMsgSetReportCode 111 amSesCreateReceiver 85 amMsgSetType 111 amSesCreateSender 86 amMsgWriteBytes 112 amSesCreateSubscriber 86 Message interface helper macros 113 amSesDelete 87 AmMsgAddStreamName 113 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeleteMessage 87 AmMsgGetStreamName 114 amSesDeletePolicy 88 Sender interface functions 115 amSesDeletePublisher 88 amSndClearErrorCodes 115 amSesDeleteReceiver 88 amSndClose 115 amSesDeleteSender 89 amSndGetCCSID 116 amSesDeleteSubscriber 89 amSndGetEncoding 116 amSesGetLastError 90 amSndGetName 117 amSesGetMessageHandle 90 amSndGopen 117	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108 amMsgSetCorrelId 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109
amSesCreatePublisher 85 amMsgSetReportCode amSesCreateReceiver 85 amMsgSetType amSesCreateSender 86 amMsgWriteBytes 86 amMsgWriteBytes 86 Message interface helper macros 86 AmMsgAddStreamName 87 AmMsgAddStreamName 887 AmMsgGetPubTimeStamp 887 AmMsgGetPubTimeStamp 888 AmMsgSetPubTimeStamp 888 Sender interface functions 888 amSesDeletePolicy 888 Sender interface functions 888 amSndClearErrorCodes 888 amSndClearErrorCodes 888 amSndClose 889 amSndGetCCSID 889 amSndGetEncoding 880 amSndGetEncoding 880 amSndGetLastError	amSesCreatePublisher 85 amMsgSetReportCode 111 amSesCreateReceiver 85 amMsgSetType 111 amSesCreateSender 86 amMsgWriteBytes 112 amSesCreateSubscriber 86 Message interface helper macros 113 amSesDelete 87 AmMsgAddStreamName 113 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeleteMessage 87 AmMsgGetStreamName 114 amSesDeletePolicy 88 Sender interface functions 115 amSesDeletePublisher 88 amSndClearErrorCodes 115 amSesDeleteReceiver 88 amSndClose 115 amSesDeleteSender 89 amSndGetCCSID 116 amSesDeleteSubscriber 89 amSndGetEncoding 116 amSesGetDistListHandle 89 amSndGetLastError 116 amSesGetLastError 90 amSndGetName 117 amSesGetMessageHandle 90 amSndOpen 117	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108 amMsgSetCorrelId 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109
amSesCreateReceiver 85 amMsgSetType amSesCreateSender 86 amMsgWriteBytes 86 amMsgWriteBytes 86 Message interface helper macros 86 AmMsgAddStreamName 87 AmMsgAddStreamName 887 AmMsgGetPubTimeStamp 888 AmMsgSetPubTimeStamp 889 Sender interface functions 888 amSndClearErrorCodes 888 amSndClearErrorCodes 888 amSndClose 889 amSndGetCCSID 889 amSndGetEncoding 880 amSndGetLastError	amSesCreateReceiver 85 amMsgSetType 111 amSesCreateSender 86 amMsgWriteBytes 112 amSesCreateSubscriber 86 Message interface helper macros 113 amSesDelete 87 AmMsgAddStreamName 113 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeleteMessage 87 AmMsgGetStreamName 114 amSesDeletePolicy 88 Sender interface functions 115 amSesDeletePublisher 88 amSndClearErrorCodes 115 amSesDeleteReceiver 88 amSndClose 115 amSesDeleteSender 89 amSndGetCCSID 116 amSesDeleteSubscriber 89 amSndGetEncoding 116 amSesGetDistListHandle 89 amSndGetLastError 116 amSesGetLastError 90 amSndGetName 117 amSesGetMessageHandle 90 amSndOpen 117	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108 amMsgSetCorrelId 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110
amSesCreateSender 86 amMsgWriteBytes Message interface helper macros Message interface helper macros AmMsgSDelete 87 AmMsgAddStreamName AmSesDeleteDistList 87 AmMsgGetPubTimeStamp AmMsgSetPubTimeStamp 88 AmMsgGetStreamName 88 Sender interface functions amSesDeletePolicy 88 Sender interface functions amSesDeletePublisher 88 amSndClearErrorCodes amSesDeleteReceiver 88 amSndClose amSesDeleteSender 89 amSndGetCCSID amSesDeleteSubscriber 89 amSndGetEncoding amSndGetLastError	amSesCreateSender 86 amMsgWriteBytes 112 amSesCreateSubscriber 86 Message interface helper macros 113 amSesDelete 87 AmMsgAddStreamName 113 amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeleteMessage 87 AmMsgGetStreamName 114 amSesDeletePolicy 88 Sender interface functions 115 amSesDeletePublisher 88 amSndClearErrorCodes 115 amSesDeleteReceiver 88 amSndClose 115 amSesDeleteSender 89 amSndGetCCSID 116 amSesDeleteSubscriber 89 amSndGetEncoding 116 amSesGetDistListHandle 89 amSndGetLastError 116 amSesGetLastError 90 amSndGetName 117 amSesGetMessageHandle 90 amSndOpen 117	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108 amMsgSetCorrelId 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110
amSesCreateSubscriber amSesDelete	amSesCreateSubscriber amSesDelete	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108 amMsgSetCorrelId 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111
amSesDelete	amSesDelete	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111
amSesDeleteDistList 87 AmMsgGetPubTimeStamp amSesDeleteMessage 87 AmMsgGetStreamName 88 Sender interface functions 88 amSesDeletePublisher 88 amSndClearErrorCodes 88 amSndClose 88 amSndClose 88 amSndClose 88 amSndClose 89 amSndGetCCSID 89 amSndGetEncoding 89 amSndGetEncoding 89 amSndGetLastError	amSesDeleteDistList 87 AmMsgGetPubTimeStamp 113 amSesDeleteMessage 87 AmMsgGetStreamName 114 amSesDeletePolicy 88 Sender interface functions 115 amSesDeletePublisher 88 amSndClearErrorCodes 115 amSesDeleteReceiver 88 amSndClose 115 amSesDeleteSender 89 amSndGetCCSID 116 amSesDeleteSubscriber 89 amSndGetEncoding 116 amSesGetDistListHandle 89 amSndGetLastError 116 amSesGetLastError 90 amSndGetName 117 amSesGetMessageHandle 90 amSndOpen 117	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112
amSesDeleteMessage 87 AmMsgGetStreamName 88 Sender interface functions 88 amSesDeletePolicy 88 amSndClearErrorCodes 88 amSndClose 88 amSndClose 89 amSndGetCCSID 89 amSesDeleteSubscriber 89 amSndGetEncoding 89 amSndGetLastError 89 amSndGetLastError	amSesDeleteMessage 87 AmMsgGetStreamName 114 amSesDeletePolicy 88 Sender interface functions 115 amSesDeletePublisher 88 amSndClearErrorCodes 115 amSesDeleteReceiver 88 amSndClose 115 amSesDeleteSender 89 amSndGetCSID 116 amSesDeleteSubscriber 89 amSndGetEncoding 116 amSesGetDistListHandle 89 amSndGetLastError 116 amSesGetLastError 90 amSndGetName 117 amSesGetMessageHandle 90 amSndOpen 117	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113
amSesDeletePolicy	amSesDeletePolicy . 88 Sender interface functions . 115 amSesDeletePublisher . 88 amSndClearErrorCodes . 115 amSesDeleteReceiver . 88 amSndClose 115 amSesDeleteSender . 89 amSndGetCCSID . 116 amSesDeleteSubscriber . 89 amSndGetEncoding . 116 amSesGetDistListHandle . 89 amSndGetLastError . 116 amSesGetLastError . 90 amSndGetName . 117 amSesGetMessageHandle . 90 amSndOpen	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 86	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113 AmMsgAddStreamName 113
amSesDeletePublisher	amSesDeletePublisher	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 86 . 86	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetForoupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113 AmMsgAddStreamName 113 AmMsgGetPubTimeStamp 113
amSesDeletePublisher88amSndClearErrorCodesamSesDeleteReceiver88amSndCloseamSesDeleteSender89amSndGetCCSIDamSesDeleteSubscriber89amSndGetEncodingamSesGetDistListHandle89amSndGetLastError	amSesDeletePublisher88amSndClearErrorCodes115amSesDeleteReceiver88amSndClose115amSesDeleteSender89amSndGetCCSID116amSesDeleteSubscriber89amSndGetEncoding116amSesGetDistListHandle89amSndGetLastError116amSesGetLastError90amSndGetName117amSesGetMessageHandle90amSndOpen117	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 86 . 86 . 87 . 87	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetElementCCSID 109 amMsgSetFormat 110 amMsgSetForoupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113 AmMsgAddStreamName 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114
amSesDeleteSender	amSesDeleteSender	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 86 . 86 . 87 . 87	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgAddStreamName 113 AmMsgAddStreamName 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115
amSesDeleteSubscriber	amSesDeleteSubscriber	Error handling	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 86 . 86 . 87 . 87 . 88	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgAddStreamName 113 AmMsgAddStreamName 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClearErrorCodes 115
amSesDeleteSubscriber	amSesDeleteSubscriber89amSndGetEncoding116amSesGetDistListHandle89amSndGetLastError116amSesGetLastError90amSndGetName117amSesGetMessageHandle90amSndOpen117	Error handling High-level functions Chapter 5. C object interface reference Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteMessage amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePublisher	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 86 . 86 . 87 . 87 . 87 . 88 . 88	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108 amMsgSetCorrelId 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgGetPubTimeStamp 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClearErrorCodes 115 amSndClose 115
amSesGetDistListHandle	amSesGetDistListHandle <td< td=""><td>Error handling High-level functions Chapter 5. C object interface reference Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteMessage amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePublisher amSesDeleteReceiver</td><td>. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 86 . 86 . 87 . 87 . 88 . 88</td><td>amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetGroupStatus 111 amMsgSetReportCode 111 amMsgSetType 111 amMsgGetPubTimeStamp 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClearErrorCodes 115 amSndClose 115 amSndGetCCSID 116</td></td<>	Error handling High-level functions Chapter 5. C object interface reference Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteMessage amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePublisher amSesDeleteReceiver	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 86 . 86 . 87 . 87 . 88 . 88	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetGroupStatus 111 amMsgSetReportCode 111 amMsgSetType 111 amMsgGetPubTimeStamp 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClearErrorCodes 115 amSndClose 115 amSndGetCCSID 116
	amSesGetLastError	Error handling High-level functions Chapter 5. C object interface reference Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteMessage amSesDeletePublisher amSesDeleteMessage amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 86 . 86 . 87 . 87 . 87 . 88 . 88 . 88	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108 amMsgSetCorrelId 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClearErrorCodes 115 amSndClose 115 amSndGetCCSID 116 amSndGetEncoding 116
	amSesGetMessageHandle	Error handling High-level functions Chapter 5. C object interface reference Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteMessage amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender amSesDeleteSubscriber	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 85 . 85 . 86 . 87 . 87 . 87 . 88 . 88 . 88 . 89 . 89	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113 AmMsgAddStreamName 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClearErrorCodes 115 amSndGetCCSID 116 amSndGetEncoding 116 amSndGetLastError 116
amSesGetMessageHandle90 amSndOpen		Error handling High-level functions Chapter 5. C object interface reference Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePolicy amSesCreateReceiver amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeleteReceiver amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 85 . 85 . 86 . 87 . 87 . 87 . 88 . 88 . 88 . 89 . 89	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113 AmMsgAddStreamName 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClearErrorCodes 115 amSndGetCCSID 116 amSndGetEncoding 116 amSndGetLastError 116 amSndGetName 117
amSesGetPolicyHandle		Error handling High-level functions Chapter 5. C object interface reference Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePolicy amSesCreateReceiver amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeletePolicy amSesDeletePolicy amSesDeletePolicy amSesDeleteReceiver amSesDeleteSender amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle amSesGetLastError	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 86 . 86 . 87 . 87 . 88 . 88 . 88 . 88 . 89 . 89 . 90	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetForupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113 AmMsgAddStreamName 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClose 115 amSndGetCCSID 116 amSndGetEncoding 116 amSndGetLastError 116 amSndGetName 117 amSndOpen 117
	amSesGetPublisherHandle 91 amSndSendFile	Error handling High-level functions Chapter 5. C object interface reference Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePolicy amSesCreateSender amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteReceiver amSesDeleteReceiver amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender amSesDeleteSender amSesDeleteSubscriber amSesDeleteSubscriber amSesGetDistListHandle amSesGetLastError amSesGetMessageHandle.	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 86 . 86 . 87 . 87 . 88 . 88 . 88 . 89 . 89 . 90 . 90	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetForupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113 AmMsgAddStreamName 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClearErrorCodes 115 amSndGetCCSID 116 amSndGetEncoding 116 amSndGetLastError 116 amSndGetName 117 amSndSend 118
amSesGetPublisherHandle 91 amSndSendFile		Error handling High-level functions Chapter 5. C object interface reference Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender amSesDeleteSender amSesDeleteSender amSesDeleteSubscriber amSesGetDistListHandle amSesGetLastError amSesGetMessageHandle amSesGetPolicyHandle	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 86 . 86 . 87 . 87 . 88 . 88 . 89 . 89 . 90 . 90 . 91	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset 107 amMsgSetCCSID 108 amMsgSetDataOffset 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetGroupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113 AmMsgAddStreamName 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClose 115 amSndGetEncoding 116 amSndGetEncoding 116 amSndGetName 117 amSndGend 118 amSndSend 118 amSndSendFile 119
amSesGetPublisherHandle 91 amSndSendFile		Error handling High-level functions Chapter 5. C object interface reference Session interface functions amSesBegin amSesClearErrorCodes amSesClose amSesCommit amSesCreate amSesCreateDistList amSesCreateMessage amSesCreatePolicy amSesCreatePublisher amSesCreateSender amSesCreateSubscriber amSesDelete amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeleteDistList amSesDeletePolicy amSesDeleteReceiver amSesDeleteReceiver amSesDeleteReceiver amSesDeleteSender amSesDeleteSender amSesDeleteSender amSesDeleteSubscriber amSesGetDistListHandle amSesGetLastError amSesGetMessageHandle amSesGetPolicyHandle	. 77 . 78 81 . 82 . 82 . 82 . 83 . 83 . 83 . 84 . 84 . 84 . 85 . 85 . 86 . 86 . 87 . 87 . 88 . 88 . 89 . 89 . 90 . 90 . 91	amMsgGetReportCode 105 amMsgGetTopic 106 amMsgGetType. 106 amMsgGetTopicCount 107 amMsgReadBytes 107 amMsgReset. 107 amMsgSetCCSID 108 amMsgSetCorrelId. 108 amMsgSetDataOffset 108 amMsgSetElementCCSID 109 amMsgSetEncoding 109 amMsgSetFormat 110 amMsgSetForupStatus 110 amMsgSetReportCode 111 amMsgSetType 111 amMsgWriteBytes 112 Message interface helper macros 113 AmMsgAddStreamName 113 AmMsgGetPubTimeStamp 113 AmMsgGetStreamName 114 Sender interface functions 115 amSndClearErrorCodes 115 amSndGetCCSID 116 amSndGetEncoding 116 amSndGetLastError 116 amSndGetName 117 amSndSend 118

amRcvBrowse	
amRcvBrowseSelect	
amRcvClearErrorCodes	
amRcvClose	
amRcvGetDefnType	4 amLibTraceText
amRcvGetLastError	5
amRcvGetName	Part 3. The C++ interface 159
amRcvGetQueueName	
amRcvOpen	6
amRcvReceive	Chapter 7. Using the Application
amRcvReceiveFile	
amRcvSetQueueName	
Distribution list interface functions	
amDstClearErrorCodes	
amDstClose	
amDstGetLastError	
amDstGetName	System default objects
amDstGetSenderCount	
amDstGetSenderCount	0 11
amDstOpen	
amDstSend	
amDstSendFile	
Publisher interface functions	
amPubClearErrorCodes	
amPubClose	
amPubGetCCSID	
amPubGetEncoding	7 Error handling
amPubGetLastError	
amPubGetName	Sending group messages
amPubOpen	8 Other considerations
amPubPublish	9 Building C++ applications
Subscriber interface functions	O AMI include files
amSubClearErrorCodes	0 C++ applications on AIX
amSubClose	
amSubGetCCSID	
amSubGetDefnType	1 C++ applications on Solaris
amSubGetEncoding	
amSubGetLastError	
amSubGetName	
amSubGetQueueName	
amSubOpen	
amSubReceive	Therper classes
amSubSetQueueName	A Exception classes
amSubSubscribe	
amSubUnsubscribe	
Policy interface functions	7
,	
amPolClearErrorCodes	
amPolGetLastError	
amPolGetName	
amPolGetWaitTime	
amPolSetWaitTime	Transactional processing
	Error handling
Chapter 6. C policy handler interface 149	
Invocation points	
AMI operation invocations	0 Set values
Pre-transport request invocations	
Post-transport request invocations	
Policy handler library functions	
amPhlCreate	
amPhlDelete	
amPhlInitialize	

AmSender		. 187	AmSession
Open and close		. 187	begin
Send			clearErrorCodes 200
Send file			close
Get values			commit
Error handling			
AmReceiver			createMessage
Open and close			createPolicy 201
Receive and browse		. 188	createPublisher 201
Receive file		. 188	createReceiver
Get values		. 188	createSender
Set value		. 188	createSubscriber 201
Error handling			deleteDistributionList 202
AmDistributionList			deleteMessage
Open and close			deletePolicy
Send			deletePublisher
Send file			
Get values			deleteSender
Error handling			deleteSubscriber 202
AmPublisher			enableWarnings 202
Open and close		. 190	getLastErrorStatus
Publish		. 190	getName
Get values		. 190	getTraceLevel
Error handling		. 190	getTraceLocation
AmSubscriber			open
Open and close			rollback
Broker messages			AmMessage
Get values			addElement
Set value			addFilter
Error handling			addTopic
AmPolicy			
Policy management			
Error handling		. 192	deleteFilter
			1.1.(.) N
Helper classes		. 193	deleteNamedElement
ÅmBytes		. 193 . 193	deleteTopic
AmBytes		. 193 . 193 . 193	deleteTopic
ÂmBytes	· · · · · · · · · · · · · · · · · · ·	. 193. 193. 193. 193	deleteTopic
ÂmBytes	· · · · · · · · · · · · · · · · · · ·	. 193. 193. 193. 193	deleteTopic
AmBytes	· · · · · · · · · · · · · · · · · · ·	. 193. 193. 193. 193. 193. 194	deleteTopic 206 enableWarnings 206 getCCSID 206 getCorrelationId 206 getDataLength 206
ÂmBytes	· · · · · · · · · · · · · · · · · · ·	. 193. 193. 193. 193. 193. 194	deleteTopic
AmBytes		. 193 . 193 . 193 . 193 . 193 . 194 . 195	deleteTopic 206 enableWarnings 206 getCCSID 206 getCorrelationId 206 getDataLength 206
AmBytes		. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195	deleteTopic 206 enableWarnings 206 getCCSID 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207
AmBytes		. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195	deleteTopic 206 enableWarnings 206 getCCSID 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207
AmBytes		. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195	deleteTopic 206 enableWarnings 206 getCCSID 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207
AmBytes		. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195	deleteTopic 206 enableWarnings 206 getCCSID 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getEncoding 207
AmBytes		. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195	deleteTopic 206 enableWarnings 206 getCCSID 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getEncoding 207 getFilter 207
AmBytes		. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 195	deleteTopic 206 enableWarnings 206 getCCSID 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getEncoding 207 getFilter 207 getFilterCount 207
AmBytes	· · · · · · · · · · · · · · · · · · ·	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 195	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFncoding 207 getFilter 207 getFilterCount 207 getFormat 207
AmBytes	· · · · · · · · · · · · · · · · · · ·	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 195 . 197	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFncoding 207 getFilter 207 getFilterCount 207 getFormat 207 getGroupStatus 208
AmBytes	e 	. 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter 207 getFilterCount 207 getFormat 207 getGroupStatus 208 getLastErrorStatus 208
AmBytes	e 	. 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 198 . 198	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter 207 getFilterCount 207 getFormat 207 getGroupStatus 208 getLastErrorStatus 208 getMessageId 208
AmBytes	e	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 198 . 198	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter 207 getFilterCount 207 getFormat 207 getGroupStatus 208 getLastErrorStatus 208 getMessageId 208 getName 208
AmBytes	e	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 198 . 198	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFncoding 207 getFilter 207 getFilterCount 207 getFormat 207 getGroupStatus 208 getLastErrorStatus 208 getMessageId 208 getName 208 getNamedElement 208
AmBytes	e 	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 198 . 198 . 198	deleteTopic 206 enableWarnings 206 getCCSID 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter 207 getFilterCount 207 getFormat 207 getGroupStatus 208 getLastErrorStatus 208 getName 208 getNamedElement 208 getNamedElementCount 208
AmBytes AmElement AmObject AmStatus AmStatus AmString Exception classes AmException AmErrorException AmErrorException AmWarningException Chapter 9. C++ interface reference Base classes Helper classes Exception classes Exception classes AmSessionFactory AmSessionFactory createSession deleteSession	e 	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 198 . 198 . 198 . 198	deleteTopic 206 enableWarnings 206 getCCSID 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter 207 getFormat 207 getFormat 207 getGroupStatus 208 getLastErrorStatus 208 getName 208 getName 208 getNamedElement 208 getReportCode 209
AmBytes	e 	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 198 . 198 . 198 . 198 . 198	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter Ount 207 getFormat 207 getFormat 207 getResageId 208 getName 208 getNamedElement 208 getNamedElementCount 208 getReportCode 209 getTopic 209
AmBytes AmElement AmObject AmStatus AmString Exception classes AmException AmErrorException AmErrorException AmWarningException Chapter 9. C++ interface reference Base classes Helper classes Exception classes AmSessionFactory AmSessionFactory createSession deleteSession getFactoryName getLocalHost getRepository	e	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 197 . 198 . 198 . 198 . 198 . 198	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter Ount 207 getFormat 207 getFormat 207 getRessageId 208 getName 208 getNamedElement 208 getNamedElementCount 208 getReportCode 209 getTopic 209 getTopicCount 209
AmBytes AmElement AmObject AmStatus AmString Exception classes AmException AmErrorException AmWarningException Chapter 9. C++ interface reference Base classes Helper classes Exception classes AmSessionFactory AmSessionFactory createSession deleteSession getFactoryName getLocalHost getRepository getTraceLevel	e	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 197 . 198 . 198 . 198 . 198 . 198	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter Ocunt 207 getFormat 207 getFormat 207 getGroupStatus 208 getLastErrorStatus 208 getName 208 getNamedElement 208 getNamedElementCount 208 getReportCode 209 getTopic 209 getTopicCount 209 getType 209
AmBytes AmElement AmObject AmStatus AmString Exception classes AmException AmErrorException AmErrorException AmWarningException Chapter 9. C++ interface reference Base classes Helper classes Exception classes AmSessionFactory AmSessionFactory createSession deleteSession getFactoryName getLocalHost getRepository getTraceLevel getTraceLocation	e	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 197 . 198 . 198 . 198 . 198 . 198 . 198	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter Ount 207 getFormat 207 getFormat 207 getRessageId 208 getName 208 getNamedElement 208 getNamedElementCount 208 getReportCode 209 getTopic 209 getTopicCount 209
AmBytes AmElement AmObject AmStatus AmString Exception classes AmException AmErrorException AmWarningException Chapter 9. C++ interface reference Base classes Helper classes Exception classes Exception classes AmSessionFactory AmSessionFactory createSession deleteSession getFactoryName getLocalHost getRepository getTraceLevel getTraceLocation setLocalHost .	e	. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 197 . 198 . 198	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter Ocunt 207 getFormat 207 getFormat 207 getGroupStatus 208 getLastErrorStatus 208 getName 208 getNamedElement 208 getNamedElementCount 208 getReportCode 209 getTopic 209 getTopicCount 209 getType 209
AmBytes AmElement AmObject AmStatus AmString Exception classes AmException AmErrorException AmErrorException AmWarningException Chapter 9. C++ interface reference Base classes Helper classes Exception classes Exception classes AmSessionFactory AmSessionFactory createSession deleteSession getFactoryName getLocalHost getRepository getTraceLevel getTraceLocation setLocalHost setRepository		. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 197 . 198 . 199 . 199 . 199 . 199	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilter Ount 207 getFormat 207 getFormat 207 getFormat 208 getLastErrorStatus 208 getName 208 getNamedElement 208 getNamedElementCount 208 getRoportCode 209 getTopic 209 getType 209 readBytes 209 reset 209
AmBytes AmElement AmObject AmStatus AmString Exception classes AmException AmErrorException AmWarningException Chapter 9. C++ interface reference Base classes Helper classes Exception classes Exception classes AmSessionFactory AmSessionFactory createSession deleteSession getFactoryName getLocalHost getRepository getTraceLevel getTraceLocation setLocalHost .		. 193 . 193 . 193 . 193 . 193 . 194 . 195 . 195 . 195 . 195 . 195 . 197 . 197 . 197 . 197 . 197 . 198 . 199 . 199 . 199 . 199 . 199	deleteTopic 206 enableWarnings 206 getCCSID. 206 getCorrelationId 206 getDataLength 206 getDataOffset 206 getElement 207 getElementCCSID 207 getElementCount 207 getFilter 207 getFilterCount 207 getFormat 207 getGroupStatus 208 getLastErrorStatus 208 getName 208 getNamedElement 208 getNamedElementCount 208 getRoportCode 209 getTopic 209 getTopicCount 209 getType 209 readBytes 209

setDataOffset			 						210	getName	. 225
setElementCCSID .									210	getQueueName	. 225
setEncoding										open	. 225
setFormat										receive	. 225
setGroupStatus										setQueueName	. 225
setReportCode										subscribe	. 226
*											. 226
J 1											
writeBytes										AmPolicy	. 227
	٠								213	clearErrorCodes	. 227
clearErrorCodes .					•				213		. 227
close									213	getLastErrorStatus	
enableWarnings .									213	getName	. 227
getCCSID									213	0-11	. 227
getEncoding									213	setWaitTime	. 227
getLastErrorStatus.									214	AmBytes	. 228
getName										•	. 228
open									214	constructors	. 228
send									214	cpy	. 229
sendFile									215	**	. 229
AmReceiver											. 229
									216	destructor	
browse	٠				•				216	length	. 229
clearErrorCodes .	•				٠				217	T	. 229
close									217	pad	. 229
enableWarnings .										AmElement	. 230
getDefinitionType .									217	AmElement	. 230
getLastErrorStatus.									218	getName	. 230
getName									218	getValue	. 230
getQueueName										getVersion	. 230
open										setVersion	. 230
receive									218		. 230
receiveFile									219	4 014	. 231
10 N									219	. '	. 231
	•	•	 •	•							
AmDistributionList .	٠	•	 •	•					220	0	. 231
clearErrorCodes .			•				-		220	getName	
close					٠				220	AmStatus	. 232
enableWarnings .										AmStatus	
get Last Error Status.									220	getCompletionCode	. 232
getName									220	getReasonCode	. 232
getSender									220	getReasonCode2	. 232
getSenderCount .								. :	220	toString	. 232
open			 						220	AmString	. 233
send										cat	. 233
sendFile									221	cmp	. 233
AmPublisher										constructors	. 233
clearErrorCodes .	•	•								contains	. 233
-1	٠	•		-		-	-	-			. 233
										17	
enableWarnings .											
getCCSID										1	. 234
getEncoding										* .	. 234
getLastErrorStatus.									222	split	. 234
getName									222	strip	. 234
open			 						223	length	. 234
publish			 						223	text	. 234
AmSubscriber										truncate	. 234
clearErrorCodes .									224	AmException	. 235
close									224	getClassName	. 235
enableWarnings .	•								224	out Committee Could	. 235
											. 235
getCCSID									224	0	
getDefinitionType .										getReasonCode	
getEncoding										getSource	
get Last Error Status.									224	toString	. 235

AmErrorException	AMHSNRQ (send request) 280
getClassName	AMHSNRS (send response) 281
getCompletionCode	AMHSB (subscribe) 282
getMethodName 236	AMHTERM (terminate) 283
getReasonCode	AMHUN (unsubscribe) 284
getSource	
toString	Chapter 12. COBOL object interface
AmWarningException	overview 285
getClassName	Session interface functions 286
getCompletionCode	Session management
getMethodName	Create objects
getReasonCode	Get object handles
getSource	Delete objects
toString	Transactional processing 287
	Error handling
Part 4. The COBOL interface 239	Message interface functions 288
	Get values
Chapter 10 Using the Application	Set values
Chapter 10. Using the Application	Reset values
Messaging Interface in COBOL 243	Read and write data
Structure of the AMI 243	Publish/subscribe topics 289
Using the repository 244	Publish/subscribe filters 289
System default objects 244	Publish/subscribe name/value elements 289
Writing applications in COBOL 246	Error handling 289
Opening and closing a session 246	Sender interface functions
Sending messages	Open and close
Receiving messages	Send
Request/response messaging	Get values
File transfer	Error handling
Publish/subscribe messaging	Receiver interface functions
Using name/value elements	Open and close
Error handling	Receive and browse
Transaction support	Get values
Sending group messages	Set values
Other considerations	Error handling
Building COBOL applications	Distribution list interface functions
COBOL applications on OS/390 257	Open and close
	Send
Chapter 11. The COBOL high-level	Get values
interface 259	Error handling
Overview of the COBOL high-level interface 260	Publisher interface functions
Initialize and terminate 260	Open and close
Sending messages	Publish
Receiving messages 260	Get values
File transfer	Error handling
Publish/subscribe	Subscriber interface functions
Transaction support	Open and close
Reference information for the COBOL high-level	Broker messages
interface	Get values
AMHBACK (backout) 263	Set value
AMHBEGIN (begin)	Error handling
AMHBRMS (browse message) 265	Policy interface functions
AMHCMIT (commit)	Get values
AMHINIT (initialize)	Set value
AMHPB (publish)	Error handling
AMHRCFL (receive file)	High-level functions
AMHRCMS (receive message)	Č
AMHRCPB (receive publication)	Chapter 13. COBOL object interface
AMHRCRQ (receive request)	reference
AMHSNFL (send file)	Session interface functions
AMHSNMS (send message) 279	bession interface functions

AMSEBG (begin)	AMMSRS (reset)
AMSECLEC (clear error codes) 300	AMMSSTCC (set CCSID)
AMSECL (close)	AMMSSTCI (set correl ID)
AMSECM (commit)	AMMSSELC (set element CCSID) 328
AMSECR (create)	AMMSSTDO (set data offset)
AMSECRDL (create distribution list) 302	AMMSSTEN (set encoding)
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) 306	AMSNGTEN (get encoding)
AMSEDLMS (delete message) 306	AMSNGTLE (get last error)
AMSEDLPO (delete policy) 307	AMSNGTNA (get name)
AMSEDLPB (delete publisher) 307	AMSNOP (open)
AMSEDLRC (delete receiver)	AMSNSN (send)
AMSEDLSN (delete sender)	AMSNSNFL (send file)
AMSEDLSB (delete subscriber) 308	Receiver interface functions
AMSEGHDL (get distribution list handle) 308	AMRCBR (browse)
AMSEGTLE (get last error codes) 309	AMRCBRSE (browse selection message) 340
AMSEGHMS (get message handle) 309	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)	AMRCGTON (get name)
AMSEGHSB (get subscriber handle) 311	AMRCGTQN (get queue name)
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) 315	AMDLCLEC (clear error codes)
AMMSCLEC (clear error codes)	AMDLCL (close)
AMMSDEEL (delete element)	AMDLGTLE (get last error)
AMMSDEFI (delete filter)	AMDLGTNA (get name)
AMMSDENE (delete named element) 316	AMDLGTSC (get sender count)
AMMSDETO (delete topic)	AMDLGTSH (get sender handle) 350
AMMSGELC (get element CCSID) 317	AMDLOP (open)
AMMSGTCC (get CCSID)	AMDLSN (send)
AMMSGTCI (get correl ID)	AMDLSNFL (send file)
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)	AMPBGTCC (get CCSID)
AMMSGTFC (get filter count)	AMPBGTEN (get encoding)
AMMSGTFI (get filter)	AMPBGTLE (get last error)
AMMSGTFO (get format)	AMPBGTNA (get name)
AMMSGTGS (get group status) 322	AMPBOP (open)
AMMSGTLE (get last error)	AMPBPB (publish)
AMMSGTMI (get message ID)	Subscriber interface functions
AMMSGTNA (get name)	AMSBCLEC (clear error codes)
AMMSGTNE (get named element) 324	AMSBCL (close)
AMMSGTNC (get named element count) 324	AMSBGTCC (get CCSID)
AMMSGTRC (get report code) 325	AMSBGTDT (get definition type)
AMMSGTTO (get topic)	AMSBGTEN (get encoding)
AMMSGTTC (get topic count)	AMSBGTLE (get last error)
AMMSGTTY (get type)	AMSBGTNA (get name)
AMMSREBY (read bytes) 327	AMSBGTON (get queue name)

AMSBOP (open)	. 361	AmSender
AMSBRC (receive)	. 361	Open and close
AMSBSTQN (set queue name)	. 362	Send
AMSBSB (subscribe)	. 362	Send file
AMSBUN (unsubscribe)		Get values
Policy interface functions		Error handling
AMPOCLEC (clear error codes)		AmReceiver
AMPOGTLE (get last error)		Open and close
AMPOGTNA (get name)		Receive and browse
AMPOGTWT (get wait time)		Receive file
AMPOSTWT (set wait time)	366	Get values
min OSIVI (Set wait time)	. 500	Set value
		Error handling
Part 5. The Java interface	367	AmDistributionList
Chapter 14. Using the Application		Open and close
	274	Send
Messaging Interface in Java		Send file
Structure of the AMI		Get values
Base classes		Error handling
Interface and helper classes		AmPublisher
Exception classes		Open and close
Using the repository		Publish
System default objects	. 372	Get values
Writing applications in Java		Error handling
Creating and opening objects	. 373	AmSubscriber
Sending messages		Open and close
Receiving messages		Broker messages
Request/response messaging		Get values
File transfer		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	. 382	AmStatus
		Exception classes
Chapter 15. Java interface overview	385	AmException
Base classes	. 385	AmErrorException
Helper classes	. 385	AmWarningException
Exception classes		
AmSessionFactory		Chapter 16. Java interface reference 399
Constructor		Base classes
Session factory management		Helper classes
Create session		Exception classes
AmSession		
		,
Session management		AmSessionFactory
Create objects		createSession
Transactional processing		getFactoryName
Error handling		getLocalHost 400
AmMessage		getRepository
Get values		getTraceLevel 400
Set values	. 388	getTraceLocation 400
Reset values	. 388	setLocalHost
Read and write data	. 388	setRepository
Publish/subscribe filters	. 389	setTraceLevel
Publish/subscribe topics		setTraceLocation
Publish/subscribe name/value elements		AmSession
Error handling		begin

clearErrorCodes					 . 402	enableWarnings	
close						getCCSID	
commit						getEncoding	
createDistributionList						getLastErrorStatus	
createMessage					 . 403	getName	
createPolicy					 . 403	open	414
createPublisher						send	414
createReceiver					 403	sendFile	415
createSender					 403	AmReceiver	416
createSubscriber					 403	browse	416
enableWarnings					 404	clearErrorCodes	417
getLastErrorStatus					 404	close	417
getName					 404	enableWarnings	417
getTraceLevel					404	getDefinitionType	417
getTraceLocation						getLastErrorStatus	
open						getName	
rollback						getQueueName	
AmMessage					405	open	
addElement					405	receive	
addFilter						receiveFile	
addTopic							419
clearErrorCodes						AmDistributionList	
deleteElement						clearErrorCodes	
deleteFilter						close	
deleteNamedElement					406	enableWarnings	
deleteTopic						getLastErrorStatus	
enableWarnings						getName	
getCCSID						getSender	
getCorrelationId						getSenderCount	
getDataLength						open	
						send	
getDataOffset						sendFile	
getElement							
getElementCount							
getEncoding						clearErrorCodes	
getFilter						close	
getFilterCount						enableWarnings	
getFormat						getCCSID	
getGroupStatus						getEncoding	
getLastErrorStatus						getLastErrorStatus	
getMessageId						getName	
getName						open	
getNamedElement						publish	
getNamedElementCount.						AmSubscriber	
getReportCode						clearErrorCodes	
getTopic						close	
getTopicCount					 . 409	enableWarnings	424
getType					 410	getCCSID	
readBytes					 410	getDefinitionType	424
reset					 410	getEncoding	424
setCCSID					 410	getLastErrorStatus	424
setCorrelationId					 410	getName	425
setDataOffset					 410	getQueueName	425
setEncoding					. 411	open	
setFormat						receive	
setGroupStatus						setQueueName	
setReportCode						subscribe	
							426
writeBytes						AmPolicy	
AmSender						clearErrorCodes	
clearErrorCodes						enableWarnings	
close						getLastErrorStatus	
	- '	 -	•	•	10	03.200.210.200.000.000	

getName	Disk space
getWaitTime 427	Operating environments 441
setWaitTime	MQSeries environment 442
AmConstants	Language compilers 442
AmElement	LDAP support 443
AmElement	Installation on AIX 444
getName	Installation 444
getValue	Setting the runtime environment 445
getVersion	Directory structure (AIX) 445
setVersion	Installation on AS/400
toString	Setting the runtime environment for Java
AmObject	programs
clearErrorCodes	Directory structure (AS/400)
getLastErrorStatus	Installation on HP-UX
getName	Installation
AmStatus	Setting the runtime environment
AmStatus	Directory structure (HP-UX)
getCompletionCode	Installation on OS/390
getReasonCode	Installation
getReasonCode2	Setting the runtime environment
toString	Unicode character conversion
AmException	Directory structure (OS/390)
getClassName	Installation on Sun Solaris
getCompletionCode	Installation
getMethodName	Setting the runtime environment
getReasonCode	Installation on Windows
toString	Installation
AmErrorException	Setting the runtime environment
getClassName	Directory structure (Windows)
getCompletionCode	Local host and repository files (AS/400, UNIX®,
getMethodName	and Windows)
getReasonCode	Default location 471
getSource	Default names 471
toString	Overriding the default location and names 471
AmWarningException 434	Local host file
getClassName	Repository file
getCompletionCode	Local host and repository files (OS/390) 473
getMethodName	Batch, RRS-batch, IMS 473
getReasonCode	CICS
getSource	Local host file 474
toString	Repository file 474
	Repository and local host caches 475
Part 6. OS/390 Subsystems 435	The AMI Administration Tool 477
· une or o o, o o o o uno o, o como o r e e e e e e	Installation 477
Chapter 17. Writing applications for	Operation
	Connecting to MQSeries
OS/390 subsystems 437	Using MQSeries Integrator Version 1 478
Writing IMS applications using AMI 437	Using MQSeries Publish/Subscribe 478
Writing CICS applications using AMI 437	Using MQSeries Integrator Version 2 478
Writing batch applications using AMI 438	Migrating to MQSeries Integrator V2 from V1
Writing RRS-batch applications using AMI 438	and MQSeries Publish/Subscribe
RRS availability	Creating default MQSeries objects
	The sample programs
Part 7. Setting up an AMI	Sample programs for AS/400, UNIX, and
installation 439	Windows
	Running the AS/400, UNIX, and Windows
Chapter 18. Installation and sample	sample programs
	Running the sample programs (OS/390) 485
programs 441	Ramining the sample programs (05/5/0) 400

The AMI policy handler sample program		Appendix A. Reason codes and LDAP	
(amtsphlr)	. 487	error codes	
Chapter 10 Defining services		Reason code: OK	
Chapter 19. Defining services,	404	Reason code: Warning	
policies, and policy handlers		Reason code: Failed	
Services, policies, and policy handlers	. 491	Reason Code: Failed (Extended C AMI functions)	
System provided definitions		LDAP error codes	. 556
System default objects	. 492		=04
Service definitions		Appendix B. Constants and structures	
Service point (sender/receiver)		The constants and structures	
Distribution list		AMB (Boolean constants)	
Subscriber		AMBRW (Browse constants)	
Publisher		AMCC (Completion codes)	
Policy definitions		AMDEF (Service and policy definitions)	
Initialization attributes		AMDT (Definition type constants)	
General attributes		AMENC (Encoding constants)	
Send attributes		AMELEM (AMI C element structure)	
Receive attributes		AMFB (Feedback codes)	
Subscribe attributes		AMFMT (Format constants)	
Publish attributes		AMGF and AMGRP (Group status constants)	
Handler attributes		AMH (Handle constants)	
Policy handler definitions		AMLEN (String length constants)	. 563
Policy handler attributes	. 504	AMMCD (Message Content Descriptor tag	
Observation 00 High topical abstractions		names)	
Chapter 20. Lightweight Directory		AMMT (Message types)	
Access Protocol support		AMPOINTER (Pointer definition)	. 563
Getting Started With LDAP	. 505	AMPS (Publish/subscribe)	. 563
SecureWay Directory	. 506	AMRC (Reason codes)	. 564
Active Directory	. 506	AMSD (System default names and handle	
Installation		synonyms)	
SecureWay Directory		AMWT (Wait time constant)	
Active Directory		C constants used by extended C AMI functions	571
Uninstallation		AMCON (Connection object integer property	 1
Updating LDAP from a repository		value constants)	. 5/1
Using the AMI Administration Tool		AMCON_INT (Connection object integer	 1
Using the command line		property identifiers)	. 5/1
Directory information tree	. 512	AMCON_STR (Connection object string	F71
Directory search		property identifiers)	
Security	. 513	AMIL (Handle constant)	
		AMH (Handle constants)	. 5/1
Chapter 21. Problem determination	515	AMH (Handle property limit constants for all object types).	E71
Using trace (AS/400, UNIX, and Windows)	. 515	AMINT (Positive integer property limit	. 371
Trace filename and directory	. 516		571
C++ and Java		constants for all object types)	
Example trace		AMLONG (Signed integer property limit	. 371
Using trace (OS/390)			572
Formatted Trace		constants for all object types)	. 372
Control of formatted trace	. 529	with amxMsgAssemble)	572
GTF Trace	. 530	AMMSG (Message object integer property valu-	
Control of GTF Trace	. 530	constants)	
When your AMI program fails	. 532	AMMSG_INT (Message object integer property	
Reason Codes		identifiers)	
First failure symptom report (AS/400, UNIX,		AMMSG_STR (Message object string property	. 573
and Windows)	. 532		572
First failure symptom report (OS/390)		identifiers)	
Other sources of information			. 3/3
Common causes of problems	. 533	AMPOL (Policy object integer property value constants)	574
A		AMPOL_INT (Policy object integer property	. 3/4
Part 8. Appendixes	535	identifiers)	574
: /		144111111111111111111111111111111111111	. 5/1

AMPOL_STR (Policy object string property	AMPHMQGET (AMI C MQGET parameter	
identifiers)	structure)	. 586
AMPROP (Integer property true/false constants) 575	AMPHMQINQ (AMI C MQINQ parameter	=0=
AMSRV (Message object integer property value	structure)	. 587
constants)	AMPHMQOPEN (AMI C MQOPEN parameter	
AMSRV_INT (Service object integer property identifiers)	structure)	. 387
AMSRV_STR (Service object string property	structure)	589
identifiers)	AMPHMQPUT1 (AMI C MQPUT1 parameter	. 560
AMSTR (Maximum string length constant for all	structure)	. 588
	AMPHMQSET (AMI C MQSET parameter	. 500
object types)	structure)	. 589
C constants and AMI parameter structures used by	Structure)	. 00.
policy handlers	Appendix C. Extended C AMI	
AMPH (Policy handler continuation codes) 577	functions	501
AMPH (Policy handler transport types) 577		
AMPH (Policy handler maximum lengths) 577	Connection object properties	
AMRO (Receive options) 577	Connection string properties	
AMPHBGN (AMI C begin parameter structure) 577	Message object properties	
AMPHCLC (AMI C close connection parameter	Message integer properties	
structure)	Message string properties	. 602
AMPHCLD (AMI C close distribution list	Policy object properties	
parameter structure) 578	Policy integer properties	. 603
AMPHCLS (AMI C close service parameter	Policy string properties	
structure)	Service object properties	
AMPHCMT (AMI C commit parameter	Service integer properties	
structure)	Service string properties	
AMPHHPM (AMI C handle poison message	Connection object functions	. 615
parameter structure)	amxConSetStringProp	. 615
AMPHOPC (AMI C open connection parameter structure)	amxConGetStringProp	. 615
AMPHOPD (AMI C open distribution list	amxConSetIntProp	
parameter structure)	amxConGetIntProp	
AMPHOPS (AMI C open service parameter	Message object functions	
structure)	amxMsgSetStringProp	
AMPHPARM (AMI parameter union) 580	amxMsgGetStringProp	
AMPHRBK (AMI C rollback parameter	amxMsgSetIntProp	
structure)	amxMsgGetIntProp	
AMPHRCS (AMI C receive from service	amxMsgAssemble	
parameter structure)	amxMsgAllocateMem	. 620
AMPHSND (AMI C send to distribution list	amxMsgUpdated	. 620
parameter structure)	amxPolSetStringProp	
AMPHSNS (AMI C send to service parameter	amxPolGetStringProp	
structure)	amxPolSetIntProp	
C constants and MQI parameter structures used by	amxPolGetIntProp	
policy handlers	Service object functions	. 623
AMPHMQBACK (AMI C MQBACK parameter	amxSrvSetStringProp	
structure)	amxSrvGetStringProp	. 623
AMPHMQBEGIN (AMI C MQBEGIN parameter	amxSrvSetIntProp	. 624
structure)	amxSrvGetIntProp	. 624
parameter structure)		
AMPHMQCMIT (AMI C MQCMIT parameter	Appendix D. Notices	625
structure)	Trademarks	
AMPHMQCONN (AMI C MQCONN parameter		
structure)	Glossary of terms and abbreviations	629
AMPHMQCONNX (AMI C MQCONNX		
parameter structure) 585	Bibliography	633
AMPHMQDISC (AMI C MQDISC parameter	MQSeries cross-platform publications	
structure)	MQSeries platform-specific publications	
•	Softcopy books.	. 634

HTML format	MQSeries information available on the Internet 635
BookManager® format	Index 637
PostScript format	indox : : : : : : : : : : : : : : : : : : :
Windows Help format	Sending your comments to IBM 651

		F		0
Ia	u		ㄷ	2
 - 3		-	_	_

Tables

1	Contain default alriants	0	C			40/
	System default objects		Service point (sender/receiver).			
2.	Object interface calls used by the high-level	10.	Distribution list			. 496
	functions	11.	Subscriber			. 496
3.	System default objects 244	12.	Publisher			. 496
4.	Object interface calls used by the high-level		Initialization attributes			
	functions	14.	General attributes			. 498
5.	The sample programs for AS/400, UNIX, and	15.	Send attributes			. 498
	Windows platforms 481	16.	Receive attributes			. 500
6.	The sample programs for OS/390 ('batch'	17.	Subscribe attributes			. 502
	includes RRS-batch) 484	18.	Publish attributes			. 503
7.	System provided definitions 492	19.	Handler attributes			. 503
	System default objects 492		Policy Handler attributes			

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 159 describes how to use the AMI in C++ programs.
- "Part 4. The COBOL interface" on page 239 describes how to write AMI programs using the COBOL high-level and object interfaces.
- "Part 5. The Java interface" on page 367 describes how to use the AMI in Java programs.
- "Part 6. OS/390 Subsystems" on page 435 gives advice on writing AMI applications for OS/390[®] subsystems.

About this book

• "Part 7. Setting up an AMI installation" on page 439 is for systems administrators who are setting up an Application Messaging Interface installation.

Conventions used in this book

This book uses the following type styles:

Format The name of a parameter in 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

The term "Windows" refers to Microsoft® Windows® 98, Windows NT 4, Windows Me, and Windows 2000, unless explicitly stated otherwise.

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 (SC34-5604-07)

The changes to the eighth edition of the Application Messaging Interface are updates to describe AMI Version 1.2 support for AS/400[®]. See the following sections:

- "Compiling and linking a policy handler on AS/400" on page 38
- "Directory structure (AS/400)" on page 449

Changes for the seventh edition (SC34-5604-06)

The changes to the seventh edition of the Application Messaging Interface are:

- The AMI is now supported in the Sun Solaris 8, Windows Me, and Windows 2000 operating environments.
- The AMI supports policy handler libraries, which are implemented in C. See:
 - "Writing policy handlers in C" on page 36
 - "Chapter 6. C policy handler interface" on page 149
 - "The AMI policy handler sample program (amtsphlr)" on page 487
 - "Chapter 19. Defining services, policies, and policy handlers" on page 491
 - "Appendix C. Extended C AMI functions" on page 591
- The AMI supports the Lightweight Directory Access Protocol (LDAP). Service, policy, and policy handler definitions can be stored and accessed across networks by using directories that support LDAP. This provides an alternative to using repository files. See:
 - "Chapter 20. Lightweight Directory Access Protocol support" on page 505
 - "LDAP error codes" on page 556
- There are further additions to the application programming interface. See:
 - "amMsgSetReportCode" on page 111 (C)
 - "amMsgSetType" on page 111 (C)
 - "setReportCode" on page 211 (C++)
 - "setType" on page 212 (C++)
 - "setReportCode" on page 411 (Java)
 - "setType" on page 412 (Java)
- Editorial changes to clarify the syntax of the following calls:
 - "amMsgSetCorrelId" on page 108 (C)
 - "amMsgSetFormat" on page 110 (C)
 - "setCorrelationId" on page 210 (C++)
 - "setFormat" on page 211 (C++)
- Updates to the installation prerequisites, the files installed, and the sample programs. See "Chapter 18. Installation and sample programs" on page 441.

Changes

- Additional reason codes. See "Reason Code: Failed (Extended C AMI functions)" on page 549.
- Additional constants. See "Appendix B. Constants and structures" on page 561.

Changes for the sixth edition (SC34-5604-05)

The changes to the sixth 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 176
 - "Building Java applications" on page 382
 - "Installation on AS/400" on page 448
 - "Local host and repository files (AS/400, UNIX®, and Windows)" on page 471
 - "Using trace (AS/400, UNIX, and Windows)" on page 515
- Editorial changes to clarify the usage notes (and some syntax) for the following calls:
 - "amBrowseMsg" on page 46 (C)
 - "amReceiveMsg" on page 53 (C)
 - "amReceiveRequest" on page 57 (C)
 - "amRcvBrowse" on page 120 (C)
 - "amRcvBrowseSelect" on page 122 (C)
 - "amRcvReceive" on page 127 (C)
 - "AMHBRMS (browse message)" on page 265 (COBOL)
 - "AMHRCMS (receive message)" on page 272 (COBOL)
 - "AMHRCRQ (receive request)" on page 276 (COBOL)
 - "AMRCBR (browse)" on page 338 (COBOL)
 - "AMRCBRSE (browse selection message)" on page 340 (COBOL)
 - "AMRCRC (receive)" on page 345 (COBOL)
- Editorial changes to clarify the syntax of the following calls:
 - "amRcvReceiveFile" on page 129 (C)
 - "AmReceiver" on page 216 (C++)
 - "AmDistributionList" on page 220 (C++)
 - "AmReceiver" on page 416 (Java)
- Editorial changes to clarify the following sections:
 - "Sample programs for AS/400, UNIX, and Windows" on page 481
 - "Service definitions" on page 494
 - "Send attributes" on page 498
 - "Receive attributes" on page 500
 - "Subscribe attributes" on page 502

Part 1. Introduction

Chapter 1. Introduction		. 3
Main components of the AMI		. 3
Sending and receiving messages		. 3
Interoperability		
Programming languages		
Description of the AMI		
Messages		
Services		
Point-to-point and publish/subscribe		
Types of service		
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.

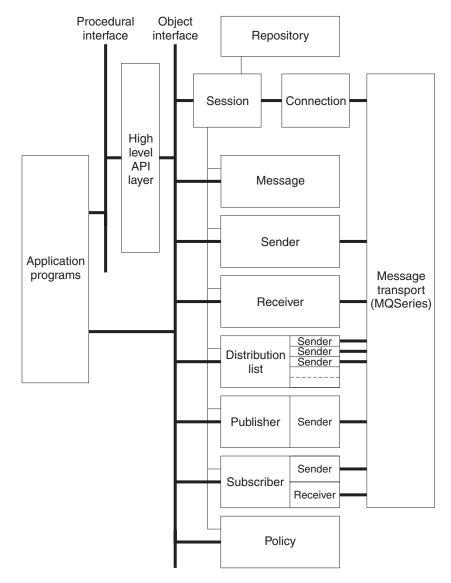


Figure 1. Basic AMI model

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 159
- "Part 4. The COBOL interface" on page 239
- "Part 5. The Java interface" on page 367

In "Part 6. OS/390 Subsystems" on page 435, you will find advice on writing AMI applications for the Information Management System (IMS $^{\text{\tiny TM}}$), Customer Information Control System (CICS) $^{\text{\tiny B}}$, batch, and RRS-batch (recoverable resource services) subsystems on OS/390.

In "Part 7. Setting up an AMI installation" on page 439, 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, Windows NT[®], Windows Me, and Windows 2000.

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	Compiling, linking and installing a policy
Interface in C	3 handler
Structure of the AMI	
Using the repository	4 AIX
System default objects	
Writing applications in C 16	
Opening and closing a session 16	6 Compiling and linking a policy handler on
Sending messages	6 HP-ŪX
Using the message object	7 Compiling and linking a policy handler on
Sample programs	8 Sun Ŝolaris
Sample programs	8 Compiling and linking a policy handler on
Using the message object	9 Windows
Sample programs	
Request/response messaging	9 Chapter 3. The C high-level interface 41
Request	Overview of the C high-level interface 42
Request. .<	0 Initialize and terminate
Sample programs	Sending messages
File transfer	
Publish/subscribe messaging	
Publish	
Subscribe	3 Transaction support
Sample programs	4 Reference information for the C high-level interface 43
Using name/value elements	
Parameters	
Example	
Event handling	5 amBrowseMsg
Error handling	5 amCommit
Transaction support	6 amInitialize
Other considerations	
Other considerations	
Multithreading	7 amReceiveMsg 53
Using MQSeries with the AMI	
Field limits	
Using the AMI OAMAS subset	
Building C applications	9 amSendMsg
AMI include file	
Data types	
Initial values for structures	9 amSubscribe 63
C applications on AIX	
Preparing C programs on AIX	
Running C programs on AIX	
C applications on AS/400	1 Chapter 4. C object interface overview 67
Preparing C programs on AS/400 31	
Running C programs on AS/400 31	
C applications on HP-UX	
Preparing C programs on HP-UX	2 Get object handles
Running C programs on HP-UX	Delete objects
C applications on OS/390	Transactional processing
Preparing C programs on OS/390	Error handling
Running C programs on OS/390 33	Message interface functions
C applications on Solaris	4 Get values
Preparing C programs on Solaris	
Running C programs on Solaris 34	
C applications on Windows	
Preparing C programs on Windows 35	
Running C programs on Windows 35	5 Publish/subscribe filters
Writing policy handlers in C	6 Publish/subscribe name/value elements
01	i ability babbelibe halife, value elements /

Error handling 71	amSesGetPublisherHandle 91
Publish/subscribe helper macros 71	amSesGetReceiverHandle
Sender interface functions	amSesGetSenderHandle
Open and close	amSesGetSubscriberHandle
Send	amSesOpen
Get values	amSesRollback
Error handling	Message interface functions
Receiver interface functions	amMsgAddElement 95
Open and close	amMsgAddFilter 95
Receive and browse	Parameters
Get values	amMsgAddTopic 96
Set values	amMsgClearErrorCodes
Error handling	amMsgDeleteElement
Distribution list interface functions	amMsgDeleteFilter
Open and close	Parameters
Send	amMsgDeleteNamedElement
Get values	amMsgDeleteTopic
Error handling	amMsgGetCCSID
Publisher interface functions	amMsgGetCorrelId
Open and close	amMsgGetDataLength99
Publish	amMsgGetDataOffset
Get values	amMsgGetElement
Error handling	amMsgGetElementCCSID
Subscriber interface functions	amMsgGetElementCount
Open and close	amMsgGetEncoding
Broker messages	amMsgGetFilter
Get values	Parameters
Set value	amMsgGetFilterCount
Error handling	Parameters
Policy interface functions	amMsgGetFormat
Get values	amMsgGetGroupStatus
Set value	amMsgGetCoroupStatus
Error handling	amMsgGetMsgId
High-level functions	amMsgGetName
Trigit-level functions	amMsgGetNamedElement
Chapter E. C. abject interfere reference	amMsgGetNamedElementCount
Chapter 5. C object interface reference 81 Session interface functions 82	amMsgGetReportCode
	amMsgGetTopic
amSesBegin	amMsgGetType
amSesClearErrorCodes	amMsgGetType
amSesClose	amMsgReadBytes
amSesCommit	· ,
amSesCreate	amMsgReset
amSesCreateDistList	amMsgSetCCSID
amSesCreateMessage	amMsgSetDataOffset
amSesCreatePolicy	amMsgSetElementCCSID
amSesCreatePublisher	
amSesCreateReceiver	amMsgSetEncoding
amSesCreateSender	amMsgSetFormat
amSesCreateSubscriber	amMsgSetGroupStatus
amSesDelete	amMsgSetReportCode
amSesDeleteDistList	amMsgSetType
amSesDeleteMessage	amMsgWriteBytes
amSesDeletePolicy	Message interface helper macros
amSesDeletePublisher	AmMsgAddStreamName
amSesDeleteReceiver	AmMsgGetPubTimeStamp
amSesDeleteSender	AmMsgGetStreamName
amSesDeleteSubscriber	Sender interface functions
amSesGetDistListHandle	amSndClearErrorCodes
amSesGetLastError	amSndClose
amSesGetMessageHandle 90	amSndGetCCSID
amSesGetPolicyHandle 91	amSndGetEncoding

amSndGetLastError amSndGetName .										. 116
amSndGetName .										. 117
amSndOpen										. 117
amSndSend										
amSndSendFile	•	•	•	•	•	•	•	•	•	. 119
Parameters Usage notes Receiver interface functi	٠	٠	•	•	•	•	•	•		. 119
Usage notes										. 119
Receiver interface functi	ion	S								. 120
amRcvBrowse										. 120
Usage notes										
amPouBroweoSoloct	•	•	•	•	•	•	•	•	•	122
amRcvBrowseSelect	•	•	•	•	•	•	•	•	•	. 122
Usage notes	٠		•	•	•	•	•	•	•	. 123
amRcvClearErrorCoc	les									. 123
Usage notes amRcvClearErrorCocamRcvClose										. 124
amRcvGetDefnType										. 124
amRcvGetLastError										
amRcvGetName .	•	·		•	•	•	•		•	. 125
annevGetName .	•	•	•	•	•	•	•	•	•	126
amRcvGetQueueNan amRcvOpen	ie	•	•	•	•	•	•	•	•	. 120
amRcvOpen			•	•	•	•	•	•	•	. 126
amRcvReceive										. 127
Usage notes										. 127
amRcvReceiveFile .										
Usage notes	•		•	•	•	•	•	•	•	. 130
amRcvSetQueueNam Distribution list interfac	ie ´	•		•	•	•	•	•	•	. 130
Distribution list interfac	e fi	unc	t101	าร						. 131
amDstClearErrorCod	.es									. 131
amDstClose										. 131
amDstGetLastError										
amDstGetName	•	Ť		•	•	•	•		•	132
amDstGetName . amDstGetSenderCou amDstGetSenderHan			•	•	•	•		•	•	122
ambsiGetSenderCou	111	•	•	•	•	•	•	•	•	. 133
amDstGetSenderHan	dle				•	•		•		. 133
amDstOpen										
amDstSend										. 134
amDstSendFile										. 135
Parameters										135
Licago notos	•	•	•	•	•	•	•	•	•	135
Parameters Usage notes Publisher interface func			•	•	•	•	•	•	•	126
Publisher interface func	t101	ns	•	•	•	•	•	•	•	. 136
amPubClearErrorCoc										
amPubClose										. 136
amPubGetCCSID .										. 136
amPubGetEncoding										. 137
amPubGetLastError										. 137
amPubGetName .										. 138
amPubOpen										. 138
amPubPublish										. 139
Subscriber interface fun-	ctic	ns								. 140
amSubClearErrorCoc	les									. 140
amSubClose										. 140
amSubGetCCSID .										. 140
amSubGetDefnType										
amSubGetEncoding										. 141
amSubGetLastError										. 142
amSubGetName .										. 142
										. 143
amSubGetQueueNan amSubOpen		•	-		-	-	-	-	-	. 143
amSubReceive										. 144
amSubSetQueueNam										. 144
amSubSubscribe .										. 145
amSubUnsubscribe										. 146
Policy interface function	ıs									. 147
amPolClearErrorCod										. 147
unin di cicui Litoi Cou		•	•	•			•	•	•	

amPolGetLastError										. 1	47
amPolGetName										. 1	48
amPolGetWaitTime											
am Pol Set Wait Time.											
Chapter 6. C policy ha	and	ler	int	erf	ac	2				1,	40
Invocation points											
AMI operation invo											
Pre-transport reques	st in	VO	cati	on	s.					. 1	51
Post-transport reque	est i	nvo	oca	tioı	าร					. 1	51
Policy handler library	func	ctio	ns							. 1	51
amPhlCreate										. 1	52
amPhlDelete											
amPhlInitialize										. 1	52
Invocation point fur	nctic	ns	(aı	nΡ	hlλ	xx) .			. 1	53
AMI extensions for pol								·	•		-
										1	
functions											
amLibRegisterFunct	ion									. 1	57
amLibTraceText										. 1	57

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.

Connection Manages the connection to the underlying message transport. This

object is not visible to the application. The AMI creates this object

implicitly and it sits beneath the session object.

Message Contains the message data, message ID, correlation ID, and options

that are used when sending or receiving a message (most of which

come from the policy definition).

Sender This is a service that represents a destination (such as 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.

Policy handler

Defines policy handler libraries, that is, collections of functions that extend the function of the AMI. These objects are not visible to the application. The AMI creates these objects implicitly, based on the

contents of the repository.

Structure of the AMI

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 Message ID), or to change the value of one or more properties before sending the message (for example, the Format). To satisfy these requirements, the AMI for C has a lower-level object style interface in addition to the high-level procedural interface. This provides access to the objects listed earlier, with methods to set and get their properties. You can mix high-level and object-level functions in the same application.

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

The high-level interface is described in "Chapter 3. The C high-level interface" on page 41. An overview of the object interface is given in "Chapter 4. C object interface overview" on page 67, with reference information in "Chapter 5. C object interface reference" on page 81.

Using the repository

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

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

System default objects

Table 1. System default objects

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

Table 1. System default objects (continued)

Default object	Constant or handle (if applicable)
	AMSD_RCV_MSG AMSD_RCV_MSG_HANDLE

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

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

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

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 78).

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 49):

```
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 64):

```
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 60). 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 70 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 107) 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 481).

Receiving messages

Use the amReceiveMsg high-level function (page 53) 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 481).

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 61) 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.)

The amReceiveRequest high-level function (page 57) 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.

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 62):

```
Sending a response message

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

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

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

Sample programs

For more details, refer to the amtshclt.c, amtshsvr.c, amtsoclt.c, and amtsosvr.c sample programs (see "Sample programs for AS/400, UNIX, and Windows" on page 481).

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 478).

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 50) 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 63) 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 65). 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 55). 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 481).

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; /* Reserved
  AMSTR
            pType;
 } AMELEM;
```

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

Parameters

i arameters	
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\ MSG\ IN\ GROUP\ for\ all\ messages\ other\ than\ first\ and\ last}
AMGRP LAST MSG IN GROUP for the last message
```

The message status is set using amMsgSetGroupStatus.

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

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 445 (AIX), page 449 (AS/400), page 455 (HP-UX), page 459 (OS/390), page 463 (Solaris), or page 468 (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 442 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 444).

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

```
ln -s /usr/mqm/lib/libamt.a /usr/lib/libamt.a
In -s /usr/mgm/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.

C applications on AIX

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
\label{limit}  \mbox{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

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 442 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 442 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 453).

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
```

C applications on HP-UX

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 442 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 442 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 461).

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, Windows NT, Windows Me, and Windows 2000 operating systems. See "Language compilers" on page 442 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.)

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

Policy handler operation is as follows:

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

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

For further details about policy handlers, invocation points, and the functions used for policy handler operation, see "Chapter 6. C policy handler interface" on page 149.

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

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

Compiling, linking and installing a policy handler

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

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

AMI extensions for policy handler callback functions. amtxc.h

amtphlc.h Policy handler interface definition.

Transport-specific policy handler definitions for MQSeries. amtphmqc.h

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

amPh1Create amPhlInitialize amPhlDelete

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

Compiling and linking a policy handler on AIX

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

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

To compile a policy handler library, first compile the object file with the non-threaded or threaded compiler using the xlc or xlc_r command with any associated compiler options. Then, use the 1d command to link the required libraries, also specifying the export file.

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

- The location of the AMI include file and other include files.
 - To do this, use the -I flag.
 - On AIX, these files are in /usr/mqm/amt/inc. Policy handlers such as the AMI policy handler sample program (amtsphlr) also use the MQI include files in /usr/mgm/tools/c/inc.
- The location of the AMI library and other libraries.
 - To do this, use the -L flag.
 - On AIX, these libraries are in /usr/mgm/lib.
- Link with the AMI library and other libraries.

To do this, use the -1 flag, more specifically -1 amt and, for the MQSeries non-threaded library, -1mqm (or -1mqmic for an MQSeries client). For a threaded policy handler library, use -lmqm r (or -lmqmic r).

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

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

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

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

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

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

Compiling and linking a policy handler on AS/400

On AS/400, policy handlers must be service programs (*SRVPGM) located in the QMQMAMIPHL library. The service program name is the library name of the policy handler, as specified using the AMI Administration Tool. The service program name does not require any prefix or suffix.

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

To do this, use the following steps:

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

Use the CRTCMOD command to create a module.

3. Create the service program.

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

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

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

Compiling and linking a policy handler on HP-UX

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

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

To compile a policy handler library, first compile the object file using the acc command with any associated compiler options. Then use the 1d command to link the required libraries, also specifying the export file.

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

• The location of the AMI include file and other include files. To do this, use the -I flag.

On HP-UX, these files are in /opt/mgm/amt/inc. Policy handlers such as the AMI

policy handler sample program (amtsphlr) also use the MQI include files in /opt/mgm/tools/c/inc.

The location of the AMI library and other libraries. To do this, use the -L flag.

On HP-UX, these libraries are in /opt/mqm/lib.

Link with the AMI library and other libraries.

To do this, use the -1 flag, more specifically -lamt_r and, for the MQSeries library, -lmqm r (or -lmqmic r for an MQSeries client).

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

```
aCC +eh -c -I/opt/mqm/inc -I/opt/mqm/amt/inc -0 +z -o amtsphlr.o amtsphlr.c
```

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

Compiling and linking a policy handler on Sun Solaris

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

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

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

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

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

To do this, use the -I flag.

On Sun Solaris, these files are in /opt/mgm/amt/inc. Policy handlers such as the AMI policy handler sample program (amtsphlr) also use the MQI include files in /opt/mqm/tools/c/inc.

• The location of the AMI library and other libraries.

To do this, use the -L flag.

On Sun Solaris, these libraries are in /opt/mqm/lib.

• Link with the AMI library and other libraries.

To do this, use the -1 flag, more specifically -1 amt and, for the MQSeries library, -lmqm (or -lmqmic for an MQSeries client).

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

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

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

Compiling and linking a policy handler on Windows

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

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

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

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

• The location of the AMI library and other libraries. To do this, include the library file amt.lib plus any other library files (e.g. mqm.lib or mqmic32.lib) as command line arguments. These library files must be

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

a directory that is specified by the lib environment variable.

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

cl -c -Ic:\mgseries\amt\include -Ic:\mgseries\tools\c\include amtsphlr.c

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

link -nod -nologo -dll amtsphlr.obj amtsphlr.exp amt.lib mgm.lib msvcrt.lib oldnames.lib kernel32.lib ws2 32.lib mswsock.lib advapi32.lib user32.lib netapi32.lib -out:amtsphlr.dll

Chapter 3. The C high-level interface

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

This chapter contains:

- "Overview of the C high-level interface" on page 42
- "Reference information for the C high-level interface" on page 43

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

amInitialize page 49 amTerminate page 64

Sending messages

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

amSendMsg page 60 amSendRequest page 61 amSendResponse page 62

Receiving messages

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

amReceiveMsg page 53 amReceiveRequest page 57 amBrowseMsg page 46

File transfer

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

amSendFile page 59 amReceiveFile page 51

Publish/subscribe

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

amPublish page 50 amSubscribe page 63 amUnsubscribe page 65 amReceivePublication page 55

Transaction support

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

amBegin page 45 amCommit page 48 amBackout page 44

Reference information for the C high-level interface

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

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

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

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

amBackout

Function to back out a unit of work.

AMBOOL amBackout(AMHSES hSession, 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 94). 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

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).

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).

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 94).

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).

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 94). 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 94). The message object is implicitly reset before

the receive takes place.

pCompCode Completion code (output).

pReason Reason code (output).

Usage notes

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

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

To return data in an application message buffer:

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

If the value of buffLen is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not

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 94).

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 94).

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).

The name of a receiver service (input). If specified as NULL, the receiverName

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).

The length of the message data, in bytes (output). Specify as NULL pDataLen

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 94). 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

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

• set pData to NULL

pDataLen that is not 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 51 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 94).

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 94).

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 94).

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 94).

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 94).

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, PAMLONG 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 94).

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 68
Message	page 70
Sender	page 72
Receiver	page 73
Distribution list	page 74
Publisher	page 75
Subscriber	page 76
Policy	page 77

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 78.

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 83
amSesOpen	page 92
amSesClose	page 83
amSesDelete	page 87

Create objects

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

am Ses Create Message	page 84
am Ses Create Sender	page 86
amSesCreateReceiver	page 85
am Ses Create Dist List	page 84
am Ses Create Publisher	page 85
am Ses Create Subscriber	page 86
amSesCreatePolicy	page 84

Get object handles

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

amSesGetMessageHandle	page 90
am Ses Get Sender Handle	page 92
am Ses Get Receiver Handle	page 91
am Ses Get Dist List Handle	page 89
am Ses Get Publisher Handle	page 91
am Ses Get Subscriber Handle	page 92
amSesGetPolicyHandle	page 91

C object interface overview

Delete objects

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

amSesDeleteMessage	page 87
am Ses Delete Sender	page 89
am Ses Delete Receiver	page 88
am Ses Delete Dist List	page 87
am Ses Delete Publisher	page 88
am Ses Delete Subscriber	page 89
amSesDeletePolicy	page 88

Transactional processing

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

amSesBeginpage 82amSesCommitpage 83amSesRollbackpage 93

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 82amSesGetLastErrorpage 90

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 98
amMsgGetCorrelId	page 98
amMsgGetElementCCSID	page 100
amMsgGetEncoding	page 100
amMsgGetFormat	page 102
am Msg Get Group Status	page 102
amMsgGetMsgId	page 103
amMsgGetName	page 104
am Msg Get Report Code	page 105
amMsgGetType	page 106

Set values

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

amMsgSetCCSID	page 108
amMsgSetCorrelId	page 108
amMsgSetElementCCSID	page 109
amMsgSetEncoding	page 109
amMsgSetFormat	page 110
am Msg Set Group Status	page 110
am Msg Set Report Code	page 111
amMsgSetType	page 111

Reset values

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

amMsgReset page 107

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 99
am Msg Get Data Off set	page 99
am Msg Set Data Off set	page 108
amMsgReadBytes	page 107

amMsgWriteBytes page 112

Publish/subscribe topics

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

amMsgAddTopicpage 96amMsgDeleteTopicpage 98amMsgGetTopicpage 106amMsgGetTopicCountpage 107

Publish/subscribe filters

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

amMsgAddFilterpage 95amMsgDeleteFilterpage 97amMsgGetFilterpage 101amMsgGetFilterCountpage 101

Publish/subscribe name/value elements

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

amMsgAddElement page 95
amMsgDeleteElement page 96
amMsgGetElement page 99
amMsgGetElementCount page 100
amMsgDeleteNamedElement page 97
amMsgGetNamedElement page 104
amMsgGetNamedElementCount page 105

Error handling

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

amMsgClearErrorCodespage 96amMsgGetLastErrorpage 103

Publish/subscribe helper macros

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

AmMsgAddStreamNamepage 113AmMsgGetPubTimestamppage 113AmMsgGetStreamNamepage 114

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.

amSndOpen page 117 amSndClose page 115

Send

Function to send a message.

amSndSend page 118 amSndSendFile page 119

Get values

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

amSndGetCCSID page 116 amSndGetEncoding page 116 amSndGetName page 117

Error handling

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

amSndClearErrorCodes page 115 amSndGetLastError page 116

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.

amRcvOpenpage 126amRcvClosepage 124

Receive and browse

Functions to receive or browse a message.

amRcvReceivepage 127amRcvReceiveFilepage 129amRcvBrowsepage 120amRcvBrowseSelectpage 122

Get values

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

amRcvGetDefnTypepage 124amRcvGetNamepage 125amRcvGetQueueNamepage 126

Set values

Function to set the queue name of the receiver service.

amRcvSetQueueName page 130

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 123amRcvGetLastErrorpage 125

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 133 amDstClose page 131

Send

Function to send a message to the distribution list.

amDstSend page 134 amDstSendFile page 135

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 132 amDstGetSenderCountpage 133 amDstGetSenderHandle page 133

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 131 amDstGetLastError page 132

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 138amPubClosepage 136

Publish

Function to publish a message.

amPubPublish page 139

Get values

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

amPubGetCCSIDpage 136amPubGetEncodingpage 137amPubGetNamepage 138

Error handling

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

amPubClearErrorCodespage 136amPubGetLastErrorpage 137

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.

amSubOpenpage 143amSubClosepage 140

Broker messages

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

amSubSubscribepage 145amSubUnsubscribepage 146amSubReceivepage 144

Get values

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

amSubGetCCSID page 140
amSubGetDefnType page 141
amSubGetEncoding page 141
amSubGetName page 142
amSubGetQueueName page 143

Set value

Function to set the queue name of the subscriber service.

amSubSetQueueName page 144

Error handling

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

amSubClearErrorCodespage 140amSubGetLastErrorpage 142

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 148 amPolGetWaitTime page 148

Set value

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

amPolSetWaitTimepage 148

Error handling

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

amPolClearErrorCodes page 147 amPolGetLastErrorpage 147

High-level functions

Each high-level function described in "Chapter 3. The C high-level interface" on page 41 calls a number of the object interface functions, as shown in the following table.

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

High-level function	Equivalent object interface calls 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 82
Message	page 94
Sender	page 115
Receiver	page 120
Distribution list	page 131
Publisher	page 136
Subscriber	page 140
Policy	page 147

Within each section the functions are listed in alphabetical order.

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

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

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

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

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

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

Session interface functions

A session object provides the scope for a unit of work and creates and manages all other objects, including at least one connection object. Each (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,
  AMHP0L
            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

pReason

Clears the error codes in the session object.

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

hSess The session handle returned by amSesCreate (input).

pCompCode Completion code (output). 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.

 $\begin{tabular}{ll} \bf pCompCode & Completion \ code \ (output). \end{tabular}$

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).

amSesCreateSender

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

AMHSND amSesCreateSender(
AMHSES hSess,
AMSTR name,

PAMLONG pCompCode, PAMLONG pReason);

hSess The session handle returned by amSesCreate (input).

name The name of the sender service (input). If it matches a sender

defined in the repository, the sender will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the sender name).

If a repository is being used and the named sender is not found in the repository, a completion code of AMCC_WARNING is returned

with a reason code of AMRC_SENDER_NOT_IN_REPOS.

pCompCode Completion code (output).

pReason Reason code (output).

amSesCreateSubscriber

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

AMHSUB amSesCreateSubscriber(

AMHSES hSess, AMSTR name, PAMLONG pCompCode, PAMLONG pReason);

hSess The session handle returned by amSesCreate (input).

name The name of the subscriber (input). If it matches a subscriber

defined in the repository, the subscriber will be created using the repository definition, otherwise it will be created with default values (that is, with a sender service name that matches the

subscriber name, and a receiver service name that is the same with

the addition of the suffix '.RECEIVER').

If a repository is being used and the named subscriber is not found

in the repository, a completion code of AMCC_WARNING is

returned with a reason code of

AMRC_SUBSCRIBER_NOT_IN_REPOS.

pCompCode Completion code (output).

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,
PAMLONG pReason);
```

hSess The session handle returned by amSesCreate (input).

phPolicy A *pointer* to the policy handle (input/output).

pCompCode Completion code (output).

pReason Reason code (output).

amSesDeletePublisher

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

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

hSess The session handle returned by amSesCreate (input).

phPub A *pointer* to the publisher handle (input/output).

pCompCode Completion code (output).

pReason Reason code (output).

amSesDeleteReceiver

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

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

hSess The session handle returned by amSesCreate (input).

phReceiver A *pointer* to the receiver service handle (input/output).

pCompCode Completion code (output).

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 107) 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 95
am Msg Delete Element	page 96
amMsgGetElement	page 99
am Msg Get Element Count	page 100
am Msg Delete Name d Element	page 97
amMsgGetNamedElement	page 104
amMsgGetNamedElementCount	

page 105

amMsgAddTopic page 96 amMsgDeleteTopic page 98 amMsgGetTopic page 106 amMsgGetTopicCount page 107

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).

 $\begin{tabular}{ll} \bf pCompCode & Completion \ code \ (output). \end{tabular}$

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).

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

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

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

hMsg The message handle returned by amSesCreateMessage (input).

pEncoding The encoding of the message (output). The following values can be

returned:

AMENC_NATIVE AMENC_NORMAL

AMENC_NORMAL_FLOAT_390 AMENC_REVERSED AMENC_REVERSED_FLOAT_390

AMENC UNDEFINED

pCompCode Completion code (output).

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

pReason

hMsg The message handle returned by amSesCreateMessage (input).

pCount The number of filters (output).

Reason code (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). In addition to application defined

values, the following can be returned:

AMFB_CODE_EXPIRATION AMFB_CODE_COA AMFB_CODE_COD AMFB_NONE

pCompCode Completion code (output).

amMsgGetTopic

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

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

hMsg The message handle returned by amSesCreateMessage (input).

topicIndex The index of the required topic in the message (input). Specifying

an index of zero returns the first topic. amMsgGetTopicCount gets

the number of topics in the message.

buffLen The length in bytes of a buffer in which the topic is returned

(input). If buffLen is specified as zero, only the topic length is

returned (in pTopicLen), not the topic itself.

pTopicLen The length of the topic, in bytes (output).

pTopic The topic (output).

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetType

Gets the message type from a message.

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

hMsg The message handle returned by amSesCreateMessage (input).

PType The message type (output). The following values can be returned:

AMMT_DATAGRAM AMMT_REQUEST AMMT_REPLY AMMT_REPORT

pCompCode Completion code (output).

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 94).

```
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). A value of

AMLEN_NULL_TERM specifies that the correlation identifier is a

string that is null terminated.

pCorrelId The correlation identifier (input). Specify as NULL (with a

correlIdLen of 0L) to set the correlation identifier to NULL.

pCompCode Completion code (output).

PReason Reason code (output).

amMsgSetDataOffset

Sets the data offset for reading or writing byte data. If the data offset is greater than the current data length, it is valid to write data into the message at that offset, but an attempt to read data will result in an error. See "amMsgReadBytes" on page 107 and "amMsgWriteBytes" on page 112.

```
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). Specify as NULL (with a formatLen of 0) to set

the format to NULL.

pCompCode Completion code (output).

pReason Reason code (output).

amMsgSetGroupStatus

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

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

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

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

hMsg The message handle returned by amSesCreateMessage (input).

status The group status (input). It can take one of the following values:

AMGRP_MSG_NOT_IN_GROUP AMGRP_FIRST_MSG_IN_GROUP AMGRP_MIDDLE_MSG_IN_GROUP AMGRP_LAST_MSG_IN_GROUP AMGRP_ONLY_MSG_IN_GROUP

pCompCode Completion code (output).

amMsgSetReportCode

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

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

hMsg The message handle returned by amSesCreateMessage (input).

code The feedback (or report code) value (input). In addition to

application defined values, the following can be set:

AMFB_CODE_EXPIRATION AMFB_CODE_COA AMFB_CODE_COD AMFB_NONE

pCompCode Completion code (output).

pReason Reason code (output).

amMsgSetType

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

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

hMsg The message handle returned by amSesCreateMessage (input).

type The message type (input). It can take one of the following values:

AMMT_DATAGRAM AMMT_REQUEST AMMT_REPLY AMMT_REPORT

pCompCode Completion code (output).

amMsgWriteBytes

Writes the specified number of data bytes into the message object, starting at the current data offset. See "amMsgSetDataOffset" on page 108.

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(hMsg, AMHMSG buffLen, AMLONG 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). Completion code (output). pCompCode

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, policies, and policy handlers" on page 491). Once created, it must not be opened until used in its correct context as a response sender when receiving a request message with **amRcvReceive** or **amReceiveRequest**. When opened, its queue and queue manager properties are modified to reflect the *ReplyTo* destination specified in the message being received. When first used in this context, the sender service becomes a response sender service.

amSndClearErrorCodes

Clears the error codes in the sender object.

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

hSender The sender handle returned by amSesCreateSender (input).

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

amSndClose

Closes the sender service.

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

hSender The sender handle returned by amSesCreateSender (input).

hPolicy The handle of a policy (input). If specified as

AMH_NULL_HANDLE, the system default policy (constant:

AMSD_POL_HANDLE) is used.

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

amSndGetCCSID

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

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

hSender The sender handle returned by amSesCreateSender (input).

pCCSID The coded character set identifier (output).

pCompCode Completion code (output).

pReason Reason code (output).

amSndGetEncoding

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

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

hSender The sender handle returned by amSesCreateSender (input).

pCompCode The encoding (output).

pCompCode Completion code (output).

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,
AMLONG buffLen,
PAMLONG pNameLen,
AMSTR pName,
PAMLONG pCompCode,
PAMLONG pReason);
```

hSender The sender handle returned by amSesCreateSender (input).

buffLen The length in bytes of a buffer in which the name is returned

(input). If specified as zero, only the name length is returned.

pNameLen The length of the name, in bytes (output). If specified as NULL,

only the name is returned.

pName The name of the sender service (output).

pCompCode Completion code (output).

pReason Reason code (output).

amSndOpen

Opens the sender service.

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

hSender The sender handle returned by amSesCreateSender (input).

hPolicy The handle of a policy (input). If specified as

AMH_NULL_HANDLE, the system default policy (constant:

AMSD_POL_HANDLE) is used.

pCompCode Completion code (output).

amSndSend

Sends a message to the destination specified by the sender service. If the sender service is not open, it will be opened (if this action is specified in the policy options).

The message data can be passed in the message object, or as a separate parameter (this means that the data does not have to be copied into the message object before sending the message, which might improve performance, especially if the message data is large).

AMBOOL amSndSend(AMHSND hSender, AMHPOL hPolicy, AMHRCV hReceiver, AMHMSG hRcvMsg, AMLONG dataLen, PAMBYTE pData, AMHMSG hSndMsg, PAMLONG pCompCode, **PAMLONG** pReason);

hSender The sender handle returned by amSesCreateSender (input).

hPolicy The handle of a policy (input). If specified as

AMH_NULL_HANDLE, the system default policy (constant:

AMSD_POL_HANDLE) is used.

hReceiver The handle of the receiver service to which the response to this

message should be sent, if the message being sent is a request message (input). Specify as AMH_NULL_HANDLE if no response

is required.

hRcvMsg The handle of a received message that is being responded to, if this

is a response message (input). Specify as AMH_NULL_HANDLE if

this is not a response message.

dataLen The length of the message data, in bytes (input). If specified as

zero, any message data will be passed in the message object

(hSndMsg).

pData The message data, if dataLen is non-zero (input).

hSndMsg The handle of a message object that specifies the properties of the

message being sent (input). If dataLen is zero, it can also contain the message data. If specified as AMH_NULL_HANDLE, the default message object (constant: AMSD_SND_MSG_HANDLE) is

used.

pCompCode Completion code (output).

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 129 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,
            hPolicy,
  AMHPOL
  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_FIRST
                                      (AMBRW_LOCK + AMBRW_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,
            hSelMsg,
 AMHMSG
 AMLONG
            buffLen,
  PAMLONG
            pDataLen,
 PAMBYTE
            pData,
 AMHMSG
            hRcvMsg.
  AMHSND
            hSender,
  PAMLONG
            pCompCode,
 PAMLONG
            pReason);
```

hReceiver The receiver handle returned by amSesCreateReceiver (input).

hPolicy. The handle of a policy (input). If specified as

AMH_NULL_HANDLE, the system default policy (constant:

AMSD_POL_HANDLE) is used.

options Options controlling the browse operation (input). Possible values

are:

AMBRW NEXT AMBRW_FIRST AMBRW CURRENT

AMBRW RECEIVE CURRENT

AMBRW DEFAULT (AMBRW NEXT)

AMBRW_LOCK_NEXT (AMBRW_LOCK + AMBRW_NEXT) AMBRW LOCK FIRST (AMBRW LOCK + AMBRW FIRST) AMBRW LOCK CURRENT (AMBRW LOCK + AMBRW CURRENT)

AMBRW UNLOCK

AMBRW RECEIVE CURRENT is equivalent to amRcvReceive for the message under the browse cursor.

Note that a locked message is unlocked by another browse or receive, even though it is not for the same message. 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 108)

before invoking the amRcvBrowseSelect function.

buffLen The length in bytes of a buffer in which the data is returned

(input).

pDataLen The length of the message data in bytes (output). If specified as

NULL, the data length is not returned.

pData The received message data (output).

hRcvMsg The handle of the message object for the received message

(output).

hSender

The handle of the response sender service that the response message must be sent to, if this is a request message (output). This sender service must be created without a repository definition, and used exclusively for sending a response. Its definition type must be AMDT_UNDEFINED (it will be set to AMDT_RESPONSE by this call).

Specify this parameter only when the

AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse

cursor.

pCompCode Completion code (output).

pReason Reason code (output).

Usage notes

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

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

To return data in an application message buffer:

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

If the value of buffLen is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an

AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a value for pDataLen that is not NULL.

To return only the data length:

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

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

amRcvClearErrorCodes

Clears the error codes in the receiver service object.

AMBOOL amRcvClearErrorCodes(
AMHRCV hReceiver,
PAMLONG pCompCode,
PAMLONG pReason);

hReceiver The receiver handle returned by amSesCreateReceiver (input).

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

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).

 $\begin{tabular}{ll} \bf pCompCode & Completion \ code \ (output). \end{tabular}$

pReason Reason code (output).

amRcvOpen

Opens the receiver service.

```
AMBOOL amRcvOpen(
AMHRCV hReceiver,
AMHPOL hPolicy,
PAMLONG pCompCode,
PAMLONG pReason);
```

hReceiver The receiver handle returned by amSesCreateReceiver (input).

The handle of a policy (input). If specified as

AMH_NULL_HANDLE, the system default policy (constant:

AMSD_POL_HANDLE) is used.

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

amRcvReceive

Receives a message.

AMBOOL amRcvReceive(AMHRCV hReceiver, hPolicy, AMHP0L 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 82) 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).

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 108) 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).

C receiver 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), 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,
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).

C distribution list interface

amDstGetLastError

Gets the information (completion and reason codes) from the last error in the distribution list object.

AMBOOL amDstGetLastError(AMHDST hDistList, AMLONG buffLen, PAMLONG pStringLen, AMSTR pErrorText, **PAMLONG** pReason2, PAMLONG pCompCode, PAMLONG pReason);

hDistList The distribution list handle returned by amSesCreateDistList

(input).

buffLen Reserved, must be zero (input). pStringLen Reserved, must be NULL (input). pErrorText Reserved, must be NULL (input).

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

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

The distribution list object name (output). pName

pCompCode Completion code (output).

amDstGetSenderCount

Gets a count of the number of sender services in the distribution list.

```
AMBOOL amDstGetSenderCount(
AMHDST hDistList,
PAMLONG pCount,
PAMLONG pCompCode,
PAMLONG pReason);
```

hDistList The distribution list handle returned by amSesCreateDistList

(input).

pCount The number of sender services (output).

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

amDstGetSenderHandle

Returns the handle (type AMHSND) of a sender service in the distribution list object with the specified index.

```
AMHSND amDstGetSenderHandle(
AMHDST hDistList,
AMLONG handleIndex,
PAMLONG pCompCode,
PAMLONG pReason);
```

hDistList The distribution list handle returned by amSesCreateDistList

(input).

handleIndex The index of the required sender service in the distribution list

(input). Specify an index of zero to return the first sender service in the list. amDstGetSenderCount gets the number of sender services

in the distribution list.

pCompCode Completion code (output).

pReason Reason code (output).

amDstOpen

Opens the distribution list object for each of the destinations in the distribution list. The completion and reason codes returned by this function call indicate if the open was unsuccessful, partially successful, or completely successful.

```
AMBOOL amDstOpen(
AMHDST hDistList,
AMHPOL hPolicy,
PAMLONG pCompCode,
PAMLONG pReason);
```

hDistList The distribution list handle returned by amSesCreateDistList

(input).

hPolicy The handle of a policy (input). If specified as

AMH_NULL_HANDLE, the system default policy (constant:

AMSD_POL_HANDLE) is used.

pCompCode Completion code (output).

C distribution list interface

amDstSend

Sends a message to each sender in the distribution list.

AMBOOL amDstSend(AMHDST hDistList, AMHPOL hPolicy, AMHRCV hReceiver AMLONG dataLen, PAMBYTE pData, hMsg, AMHMSG **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).

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 129 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).

amPubGetEncoding

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

```
AMBOOL amPubGetEncoding(
   AMHPUB hPublisher,
   PAMLONG pEncoding,
   PAMLONG pCompCode,
   PAMLONG pReason);

hPublisher The publisher handle returned by amSesCreatePublisher (input).

pEncoding The encoding (output).
```

pCompCode Completion code (output).

pReason Reason code (output).

amPubGetLastError

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

```
AMBOOL amPubGetLastError(
AMHPUB hPublisher,
AMLONG buffLen,
PAMLONG pStringLen,
AMSTR pErrorText,
PAMLONG pReason2,
PAMLONG pCompCode,
PAMLONG pReason);
```

hPublisher The publisher handle returned by **amSesCreatePublisher** (input).

buffLen Reserved, must be zero (input).pStringLen Reserved, must be NULL (input).pErrorText Reserved, must be NULL (input).

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

NULL. If pReason indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, pReason2 gives 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

 $am Pub Get Last Error \ \ \text{function call has itself detected an error and}$

failed.

C publisher interface

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).

amPubPublish

Publishes a message using the publisher service.

The message data is passed in the message object. There is no option to pass it as a separate parameter as with **amSndSend** (this would not give any performance improvement because the MQRFH header has to be added to the message data before publishing it).

```
AMBOOL amPubPublish(
AMHPUB hPublisher,
AMHPOL hPolicy,
AMHRCV hReceiver,
AMHMSG hPubMsg,
PAMLONG pCompCode,
PAMLONG pReason);
```

hPublisher The publisher handle returned by amSesCreatePublisher (input).

hPolicy The handle of a policy (input). If specified as

AMH_NULL_HANDLE, the system default policy (constant:

AMSD_POL_HANDLE) is used.

hReceiver The handle of the receiver service to which the response to this

publish request should be sent (input). Specify as

AMH_NULL_HANDLE if no response is required. This parameter is mandatory if the policy specifies implicit registration of the

publisher.

hPubMsg The handle of a message object for the publication message (input).

If specified as AMH_NULL_HANDLE, the default message object

(constant: AMSD_SND_MSG_HANDLE) is used.

pCompCode Completion code (output).

Subscriber interface functions

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

amSubClearErrorCodes

Clears the error codes in the subscriber object.

```
AMBOOL amSubClearErrorCodes(
AMHSUB hSubscriber,
PAMLONG pCompCode,
PAMLONG pReason);
```

hSubscriber The subscriber handle returned by **amSesCreateSubscriber** (input).

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

amSubClose

Closes the subscriber service.

```
AMBOOL amSubClose(

AMHSUB hSubscriber,

AMHPOL hPolicy,

PAMLONG pCompCode,

PAMLONG pReason);
```

hSubscriber The subscriber handle returned by **amSesCreateSubscriber** (input).

hPolicy The handle of a policy (input). If specified as

AMH_NULL_HANDLE, the system default policy (constant:

AMSD_POL_HANDLE) is used.

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

amSubGetCCSID

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

```
AMBOOL amSubGetCCSID(
AMHSUB hSubscriber,
PAMLONG pCCSID,
PAMLONG pCompCode,
PAMLONG pReason);
```

hSubscriber The subscriber handle returned by **amSesCreateSubscriber** (input).

pccsid The coded character set identifier (output).

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

amSubGetDefnType

Gets the definition type of the subscriber's receiver service.

```
AMBOOL amSubGetDefnType(
AMHSUB hSubscriber,
PAMLONG pType,
PAMLONG pCompCode,
PAMLONG pReason);
```

hSubscriber The subscriber handle returned by **amSesCreateSubscriber** (input).

pType The definition type (output). It can be:

AMDT_UNDEFINED AMDT_TEMP_DYNAMIC AMDT_DYNAMIC AMDT_PREDEFINED

pCompCode Completion code (output).

pReason Reason code (output).

amSubGetEncoding

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

```
AMBOOL amSubGetEncoding(
AMHSUB hSubscriber,
PAMLONG pEncoding,
PAMLONG pCompCode,
PAMLONG pReason);
```

hSubscriber The subscriber handle returned by **amSesCreateSubscriber** (input).

pEncoding The encoding (output).pCompCode Completion code (output).

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.

Reason code (output). Not returned if specified as NULL. A value pReason

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).

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 94).

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(
            hSubscriber,
 AMHSUB
 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.

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.

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).

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 94) 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).

C subscriber interface

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 94) 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(
AMHPOL hPolicy,
PAMLONG pCompCode,
PAMLONG pReason);
```

hPolicy The policy handle returned by amSesCreatePolicy (input).

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

amPolGetLastError

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

```
AMBOOL amPolGetLastError(
AMHPOL hPolicy,
AMLONG buffLen,
PAMLONG pStringLen,
AMSTR pErrorText,
PAMLONG pReason2,
PAMLONG pCompCode,
PAMLONG pReason);
```

hPolicy The policy handle returned by **amSesCreatePolicy** (input).

bufflen Reserved, must be zero (input).pStringlen Reserved, must be NULL (input).pErrorText Reserved, must be NULL (input).

pReason2 A secondary reason code (output). Not returned if specified as NULL. If pReason indicates AMRC_TRANSPORT_WARNING or

AMRC_TRANSPORT_ERR, pReason2 gives 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.

C policy interface

amPolGetName

Returns the name of the policy object.

```
AMBOOL amPolGetName(
AMHPOL hPolicy,
AMLONG buffLen,
PAMLONG pNameLen,
AMSTR pName,
PAMLONG pCompCode,
PAMLONG pReason);
```

hPolicy The policy handle returned by amSesCreatePolicy (input).

buffLen The length in bytes of a buffer into which the name is put (input).

Set it to zero to return only the name length.

pNameLen The length of the name, in bytes (output). Set it to NULL to return

only the name.

pName The policy object name (output).

pCompCode Completion code (output).

pReason Reason code (output).

amPolGetWaitTime

Returns the wait time (in ms) set for this policy.

```
AMBOOL amPolGetWaitTime(
AMHPOL hPolicy,
PAMLONG pWaitTime,
PAMLONG pCompCode,
PAMLONG pReason);
```

hPolicy The policy handle returned by amSesCreatePolicy (input).

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).

Chapter 6. C policy handler interface

Policy handler libraries are collections of functions that are implemented in C and that are:

- · external to the AMI
- called from predefined AMI invocation points

The policy handler framework allows the AMI to be extended to include additional message handling functions that are beyond those that are in the product. Examples of such additional functions include:

- · writing audit records to a database or audit queue
- · adding additional information to the message, for example, inserting headers
- · compressing message data
- enforcing corporate messaging standards
- interfacing to a system management tool to report error conditions
- sending or receiving messages using alternative transport protocols

Vendors might provide policy handlers that are generally applicable, or customers might create policy handlers for their own specific requirements.

A policy handler is invoked when a policy used on an AMI function call is configured to specify the policy handler. More than one policy handler may be invoked on a single AMI call, and they are invoked in the order specified in the policy repository definition.

A policy handler can be called at various points during AMI processing. When the policy handler is initialized, it specifies to the AMI the list of invocation points at which it should be called.

Static initialization and invocation parameter strings can be specified for the policy handler in the repository. An additional parameter string, for use by the policy handler, can be specified for a service point.

Existing application programs that already use policy definitions in the repository (rather than a system default policy) can exploit policy handler library functions. To do this, you simply update the AMI repository by using the AMI Administration Tool. You do not need to change the application program.

To define a policy handler, you use the AMI Administration Tool to associate an administrator-defined name with the file name of the policy handler library. In an existing policy object definition, you can specify the list of policy handlers to associate with this policy. To do this, in the AMI Administration Tool, you access the policy and its Handlers tab, then select from the list of existing policy handlers (defined using the policy handler definitions pane).

A policy handler library can create and manage its own state information in a library context object. To do this, it returns a context handle on creation that is passed to it on each subsequent call. During initialization, the library must register each invocation point it wishes to support in its implementation.

C policy handler interface

Note the following features of policy handlers:

- There can be more than one policy handler library definition in the repository.
- It is possible to specify a sequence of different policy handler invocations in a policy for a single invocation point.
- It is possible to specify three separate static parameter strings in the repository
 that can be passed to the policy handler to further customize its operation.
 One parameter string is passed on initialization, and the other two are passed at
 each invocation point. These can provide information that is specific to the
 current policy, and information that is specific to the current sender or receiver
 service object.

The next sections describe invocation points and the functions required for policy handler operation. For further information about policy handlers, see:

- "Writing policy handlers in C" on page 36
- "Appendix C. Extended C AMI functions" on page 591
- "Policy definitions" on page 497 and "Policy handler definitions" on page 504
- "The AMI policy handler sample program (amtsphlr)" on page 487

Invocation points

Policy handler invocation points are predefined AMI exits, from where you can optionally invoke a policy handler function. Invocation points are provided at two levels:

- AMI operation invocation
- Transport invocation

For each AMI operation, there is a single AMI operation invocation point plus one or more transport invocation points. For example, if you open a session, this opens a connection that has an open connection (AMI) invocation point, and MQCONNX, MQOPEN, and MQINQ (transport) invocation points.

AMI operation invocations

AMI operation invocations are calls that occur at the start of each AMI processing operation that involves any transport request. Typically, these might be used to replace the existing (MQSeries) transport implementation with an alternative transport implementation. Therefore, they allow you to provide alternative pluggable transport drivers alongside MQSeries. The supported set of AMI operation invocation points are as follows:

```
Open Connection
                           (AMINV CONNECTION OPEN)
Close Connection
                           (AMINV CONNECTION CLOSE)
                           (AMINV BEGIN)
Begin
Rollback
                           (AMINV ROLLBACK)
Commit
                           (AMINV COMMIT)
Open Sender
                           (AMINV_SENDER_OPEN)
                          (AMINV_SENDER_CLOSE)
(AMINV_SENDER_SEND)
Close Sender
Send To Sender
                          (AMINV RECEIVER OPEN)
Open Receiver
                          (AMINV RECEIVER_CLOSE)
Close Receiver
Receiver From Receiver (AMINV RECEIVER RECEIVE)
Open Distribution List (AMINV DIST LIST OPEN)
Close Distribution List (AMINV DIST LIST CLOSE)
Send To Distribution List (AMINV_DIST_LIST_SEND)
Handle Poison Message
                          (AMINV HANDLE POISON MSG)
```

Pre-transport request invocations

Pre-transport request invocations are calls that occur immediately before each underlying MQSeries transport call. These calls are as follows:

```
Pre-MQBACK (AMINV_PRE_MQBACK)
Pre-MQBEGIN (AMINV_PRE_MQBEGIN)
Pre-MQCLOSE (AMINV_PRE_MQCLOSE)
Pre-MQCMIT (AMINV_PRE_MQCMIT)
Pre-MQCON (AMINV_PRE_MQCONN)
Pre-MQCONNX (AMINV_PRE_MQCONNX)
Pre-MQDISC (AMINV_PRE_MQDISC)
Pre-MQGET (AMINV_PRE_MQBET)
Pre-MQINQ (AMINV_PRE_MQINQ)
Pre-MQOPEN (AMINV_PRE_MQPEN)
Pre-MQPUT (AMINV_PRE_MQPUT)
Pre-MQPUT1 (AMINV_PRE_MQPUT1)
Pre-MQSET (AMINV_PRE_MQPUT1)
```

Post-transport request invocations

Post-transport request invocations are calls that occur immediately after each underlying MQSeries transport call. These calls are as follows:

```
Post-MQBACK (AMINV_POST_MQBACK)
Post-MQBEGIN (AMINV POST MQBEGIN)
Post-MQCLOSE (AMINV POST MQCLOSE)
Post-MQCMIT (AMINV POST MQCMIT)
Post-MQCON (AMINV_POST_MQCONN)
Post-MQCONNX (AMINV POST MQCONNX)
Post-MQDISC (AMINV_POST_MQDISC)
Post-MQGET (AMINV POST MQGET)
Post-MQINQ
           (AMINV POST MQINQ)
Post-MQOPEN (AMINV_POST_MQOPEN)
Post-MQPUT
             (AMINV POST_MQPUT)
Post-MQPUT1 (AMINV_POST_MQPUT1)
             (AMINV POST MQSET)
Post-MQSET
```

Policy handler library functions

A policy handler library must implement the following functions:

- "amPhlCreate" on page 152
- "amPhlDelete" on page 152
- "amPhlInitialize" on page 152
- "Invocation point functions (amPhlXxx)" on page 153

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

AMCC_OK Function completed successfully

AMCC_WARNING Function completed with a warning

AMCC_FAILED An error occurred during processing

The implementor of the policy handler defines the reason code values that are returned by a specific policy handler library. The value returned must be AMRC_NONE with a completion code of AMCC_OK, or non-zero with a completion code of AMCC_WARNING or AMCC_ERROR.

Policy handler library functions

amPhlCreate

Creates a library context (data area) instance and returns its handle. This function is called at session creation time. There is one library context object for a session for each policy handler library definition in the repository.

amPhlCreate returns the library context handle (with a completion code of AMCC_OK or AMCC_WARNING). If there is an error, amPhlCreate returns AMH_NULL_HANDLE with a completion code of AMCC_FAILED.

```
AMHPHL AMENTRY amPhlCreate(
  AMLONG
                                 /* Reserved, must be zero */
               type,
                                 /* Name length in chars */
  AMI ONG
               nameLen.
  AMSTR
               name,
                                 /* Name */
  PAMLONG
               pCompCode,
                                 /* Completion code */
  PAMLONG
               pReason);
                                 /* Reason code */
```

Reserved, must be zero. type

nameLen The length of the name in bytes, excluding any terminating null

(input). A value of AMLEN_NULL_TERM specifies that the name

is a null-terminated string.

name The policy handler name that the administrator assigned to this

library in the repository (input).

pCompCode Completion code (output).

pReason Policy handler reason code (output).

amPhIDelete

Deletes the specified instance of the library context.

amPhlDelete returns AMB TRUE for success (that is, with a completion code of AMCC_OK or AMCC_WARNING), or AMB_FALSE if there is an error (that is, with a completion code of AMCC_FAILED).

```
AMBOOL AMENTRY amPhlDelete(
   PAMHPHL
                                 /* Context handle */
               phContext,
   PAMLONG
               pCompCode,
                                 /* Completion code */
   PAMLONG
               pReason);
                                 /* Reason code */
```

A pointer to the library context handle (input). This should be set phContext

to AMH_INVALID_HANDLE after the policy handler has been

deleted and before returning from this function.

pCompCode Completion code (output).

pReason Policy handler reason code (output).

amPhlInitialize

Initializes the policy handler library. Before returning, this function should call back into the AMI using amLibRegisterFunction to register each of the invocation points that the policy handler wishes to support in its implementation.

amPhlInitialize returns AMB_TRUE for success (that is, with a completion code of AMCC OK or AMCC WARNING), or AMB FALSE if there is an error (that is, with a completion code of AMCC_FAILED).

Policy handler library functions

```
AMBOOL AMENTRY amPhlInitialize(
                               /* Context handle */
  AMHPHL
              hContext,
  AMHLIB
              hLibrary,
                               /* AMI library handle */
                              /* Initialization parameter length */
  AMLONG
              initParmLen,
  AMSTR
              initParms,
                              /* Initialization parameters */
  AMB00L
              traceOn,
                              /* Tracing on flag */
                              /* Completion code */
  PAMLONG
              pCompCode,
  PAMLONG
                               /* Reason code */
              pReason);
```

hContext The library context handle (input).

hLibrary The AMI library handle to be used with amLibRegisterFunction (input).

initParmLen The length of the initialization parameter string that is defined in the repository for this policy handler (excluding any terminating

null).

initParms The initialization parameter string that is defined in the repository

definition for this policy handler (input). The string is

NULL-terminated.

traceOn Set tracing on (input). This must be specified as AMB_TRUE or

AMB_FALSE.

If tracing is on, the policy handler should call back into the AMI by using amLibTraceText to add trace entries to the AMI trace

buffer.

Trace entry and exit points should be included at the start and end of each function. They should also be included at any other critical point in the code where additional information may help in problem diagnosis. Examples include before and after calls to other components.

Trace strings should include any information that may be useful for problem diagnosis, including information about error codes and return codes.

pCompCode Completion code (output).

pReason Policy handler reason code (output).

Invocation point functions (amPhIXxx)

A policy handler library must implement one or more invocation point functions, with the following function prototype, for those functions that the policy handler library implements.

A policy handler can use the same function for multiple invocation points, or use a separate function for each.

The implementor of the policy handler library determines the function names. These function names are defined by, and unique to, an individual policy handler library.

amPhlXxx returns AMB_TRUE for success (that is, with completion code AMCC_OK or AMCC_WARNING), or AMB_FALSE if there is an error (that is, with completion code AMCC_FAILED).

Policy handler library functions

```
AMBOOL AMENTRY amPh1Xxx(
  AMHPHL
             hContext,
                             /* Context handle */
            transportLen, /* Transport length in bytes */
transport, /* Transport */
  AMLONG
         transport, /* Transport */
invocationPoint, /* Invocation point (AMINV value) */
  AMSTR
  AMLONG
  PAMLONG
                            /* Continuation code */
             pContinue,
             pCompCode,
  PAMLONG
                            /* Completion code */
                             /* Reason code */
  PAMLONG
             pReason);
```

hContext The library context handle (input).

The length of the transport name in bytes. This must be 9 (see transportLen AMPH_TRANSPORT_LEN_MQ in the C header file amtphlc.h).

The transport name. This must be "MQSeries" (see transport AMPH_TRANSPORT_TYPE_MQ in the C header file amtphlc.h).

invocationPoint

The invocation point, specified as one of the AMINV_ values listed in "Invocation points" on page 150 (input).

customParmLen The length of the custom parameter string that is defined in the repository for this service point, excluding any terminating null (input).

customParms The custom parameter string that is defined in the repository for this service point (input). The string is NULL-terminated.

invParmLen The length of the invocation parameter string that is defined in the repository for this policy, excluding any terminating null (input).

invParms The invocation parameter string that is defined in the repository for this policy (input). The string is NULL-terminated.

amiOperation The current AMI operation from which this invocation originates (input). This can be one of:

> AMOP CONNECTION OPEN: open transport connection AMOP CONNECTION CLOSE: close transport connection begin unit of work roll back unit of work AMOP BEGIN: AMOP ROLLBACK: commit unit of work AMOP_COMMIT: AMOP_SENDER_OPEN: open sender
> AMOP_SENDER_CLOSE: close sender
> AMOP_SENDER_SEND: send to sender
> AMOP_RECEIVER_OPEN: open receiver AMOP RECEIVER CLOSE: close receiver AMOP RECEIVER RECEIVE: receive from receiver AMOP DIST LIST OPEN: open distribution list AMOP_DIST_LIST_CLOSE: close distribution list AMOP DIST LIST SEND: send to distribution list AMOP HANDLE POISON MSG: handle poison message

amiOperation defines the information specified by the pAmiParms parameters below, and can be used to tailor the invocation point processing to the particular AMI operation. For example, AMINV_PRE_MQOPEN with AM_SENDER_OPEN might require different processing from AMINV_PRE_MQOPEN with AMOP_RECEIVER_OPEN.

Policy handler library functions

pAmiParms

The parameters of the current AMI operation (input). This is a pointer to a union, where the contents of that union represent a structure that is determined by the **amiOperation** parameter, as follows:

AMI Operation	Structure
AMOP_CONNECTION_OPEN:	AMPHOPC
AMOP CONNECTION CLOSE:	AMPHCLC
AMOP BEGIN:	AMPHBGN
AMOP ROLLBACK:	AMPHRBK
AMOP COMMIT:	AMPHCMT
AMOP SENDER OPEN:	AMPHOPS
AMOP SENDER CLOSE:	AMPHCLS
AMOP SENDER SEND:	AMPHSNS
AMOP_RECEIVER_OPEN:	AMPHOPS
AMOP RECEIVER CLOSE:	AMPHCLS
AMOP RECEIVER RECEIVE:	AMPHRCS
AMOP_DIST_LIST_OPEN:	AMPHOPD
AMOP_DIST_LIST_CLOSE:	AMPHCLD
AMOP_DIST_LIST_SEND:	AMPHSND
AMOP_HANDLE_POISON_MSG:	AMPHHPM

pTrpParms

The parameters of the current transport operation for the invocation point (input). This is a pointer to a union, where the contents of that union represent a structure that is determined by the **invocationPoint** parameter as follows:

Structure

AMINV_CONNECTION_OPEN: None (NULL pointer) AMINV CONNECTION CLOSE: None (NULL pointer)

Invocation point

```
None (NULL pointer)
AMINV BEGIN:
                            None (NULL pointer)
                            None (NULL pointer)
AMINV ROLLBACK:
                            None (NULL pointer)
AMINV_COMMIT:
AMINV_SENDER_OPEN:
                            None (NULL pointer)
AMINV_SENDER_CLOSE: AMINV_SENDER_SEND:
                            None (NULL pointer)
                            None (NULL pointer)
AMINV RECEIVER_OPEN:
                            None (NULL pointer)
AMINV RECEIVER CLOSE:
                            None (NULL pointer)
AMINV RECEIVER RECEIVE:
                            None (NULL pointer)
AMINV DIST LIST OPEN:
                            None (NULL pointer)
                            None (NULL pointer)
AMINV_DIST_LIST_CLOSE:
AMINV_DIST_LIST_SEND:
                            None (NULL pointer)
AMINV HANDLE POISON MSG:
                            None (NULL pointer)
AMINV PRE MQBACK:
                            AMPHMQBACK
AMINV POST MQBACK:
                            AMPHMQBACK
AMINV PRE MQBEGIN:
                            AMPHMQBEGIN
AMINV POST MQBEGIN:
                            AMPHMOBEGIN
AMINV PRE MQCLOSE:
                            AMPHMQCLOSE
AMINV POST MQCLOSE:
                            AMPHMQCLOSE
AMINV PRE MQCMIT:
                            AMPHMQCMIT
AMINV_POST_MQCMIT:
                            AMPHMQCMIT
AMINV_PRE_MQCONN:
                            AMPHMQCONN
AMINV POST MQCONN:
                            AMPHMQCONN
AMINV PRE MQCONNX:
                            AMPHMQCONNX
AMINV_POST_MQCONNX:
                            AMPHMQCONNX
AMINV PRE MQDISC:
                            AMPHMODISC
AMINV POST MQDISC:
                            AMPHMQDISC
AMINV PRE MQGET:
                            AMPHMQGET
AMINV POST MQGET:
                            AMPHMQGET
AMINV_PRE_MQINQ:
                            AMPHMQINQ
AMINV_POST_MQINQ:
                            AMPHMQINQ
AMINV_PRE_MQOPEN:
                            AMPHMQOPEN
AMINV POST MOOPEN:
                            AMPHMQOPEN
AMINV PRE MOPUT:
                            AMPHMQPUT
AMINV_POST_MQPUT:
                            AMPHMQPUT
```

Policy handler library functions

AMINV_PRE_MQPUT1: AMPHMQPUT1
AMINV_POST_MQPUT1: AMPHMQPUT1
AMINV_PRE_MQSET: AMPHMQSET
AMINV_POST_MQSET: AMPHMQSET

pContinue

Returns the continuation code as one of:

AMPH CONTINUE:

Continue processing. The AMI continues with normal processing on return from the policy handler function call. If the invocation returns AMB_FALSE, indicating that the request failed, subsequent AMI processing is limited to normal error handling. With this continuation code, any output information (except for error information) that is returned via the call parameters is ignored.

AMPH COMPLETE:

Invocation is complete. The behavior differs, depending on the context of the invocation:

• Context: AMI Operation Invocation

This continuation code indicates that the policy handler has completed all processing for the current AMI operation. Subsequent processing is limited to error handling only.

• Context: Pre-Transport Invocation

This continuation code indicates that the policy handler has completed all processing for the MQSeries transport request for the current invocation point. The AMI skips the transport request and continues with post-transport invocation, where applicable. If the policy handler returns AMB_FALSE, indicating that the request failed, subsequent AMI processing is limited to normal error handling.

Context: Post-transport AMI function invocation
 The continuation code is ignored. The AMI continues with normal processing. If the policy handler returns AMB_FALSE, indicating that the request failed, subsequent AMI processing is limited to normal error handling.

pCompCode Completion code (output).

pReason Policy handler reason code (output).

AMI extensions for policy handler callback functions

The following functions represent the AMI extensions that are provided to enable the policy handler to make the necessary AMI callback requests.

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

AMCC_OK Function completed successfully

AMCC_WARNING Function completed with a warning

AMCC_FAILED An error occurred during processing

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

amLibRegisterFunction

This function is called during **amPhIInitialize** processing to register an invocation point function.

```
AMBOOL AMENTRY amLibRegisterFunction(

AMHLIB hLibrary, /* Object handle */

AMLONG invocationPoint, /* Invocation point */

PAMFUNC pFunction, /* Function pointer */

PAMLONG pCompCode, /* Completion code */

PAMLONG pReason); /* Reason code */
```

amLibRegisterFunction returns AMB_TRUE for success, or AMB_FALSE if there is an error.

hLibrary The AMI library handle parameter used with **amPhlInitialize**.

invocationPoint

The invocation point for this function pointer, specified as one of the AMINV_ values listed in "Invocation points" on page 150 (input).

pFunction The function address for this invocation point.

pCompCode Completion code (output).

pReason AMI Reason code (output).

amLibTraceText

This function traces the specified text string and adds it to the AMI trace buffer if tracing is on. If tracing is off, it does nothing.

```
AMBOOL AMENTRY amLibTraceText(
  AMLONG
               control,
                                /* Trace control (AMTC value) */
  AMLONG
               stringLength,
                               /* Text string length */
                               /* Trace text string */
  AMSTR
               string,
  PAMLONG
               pCompCode,
                               /* Completion code */
                                /* Reason code */
  PAMLONG
               pReason);
```

control Control information, which can be one of the following values:

AMTC_TEXT:

Trace text string.

AMTC FUNCTION ENTRY:

Trace function entry (the function name should be included in the text string parameter).

AMI policy handler extensions

AMTC_FUNCTION_EXIT:

Trace function exit (the function name and any return code value should be included in the text string parameter).

AMTC_DEFAULT:

The same as AMTC_TEXT, that is, trace text string.

stringLength The length of the text string name in bytes (excluding any

terminating null).

pCompCode The text string to be traced.

Completion code (output).

pReason AMI Reason code (output).

Part 3. The C++ interface

Chapter 7. Using the Application Messaging	Create objects
Interface in C++	Delete objects
Structure of the AMI	Transactional processing
Base classes	Error handling
Interface and helper classes 164	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 186
Sample program	Error handling
Receiving messages	AmSender
Sample program	Open and close
Request/response messaging 168	Send
Sample programs	Send file
File transfer	Get values
Publish/subscribe messaging 170	Error handling
Sample programs 171	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	AmDistributionList
Field limits	Open and close
Building C++ applications	Send
AMI include files	Send file
	Get values
C++ applications on AIX	
Preparing C++ programs on AIX 175	Error handling
Running C++ programs on AIX 176	AmPublisher
C++ applications on AS/400	Open and close
Preparing C++ programs on AS/400 176	Publish
Running C++ programs on AS/400 177	Get values
C++ applications on HP-UX	Error handling
Preparing C++ programs on HP-UX 177	AmSubscriber
Running C++ programs on HP-UX 177	Open and close
C++ applications on Solaris	Broker messages
Preparing C++ programs on Solaris 178	Get values
Running C++ programs on Solaris 178	Set value
C++ applications on Windows 179	Error handling
Preparing C++ programs on Windows 179	AmPolicy
Running C++ programs on Windows 179	Policy management
	Error handling
Chapter 8. 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

Chapter 9. C++ interface reference			197	getFilter	207
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				getType	
setLocalHost				readBytes	
setRepository				reset	
setTraceLevel			199	setCCSID	
setTraceLocation			199	setCorrelationId	210
AmSession			200	setDataOffset	210
begin			200	setElementCCSID	210
clearErrorCodes				setEncoding	210
close			200	setFormat	
commit			200	setGroupStatus	
createDistributionList			200	setReportCode	
createMessage				setType	
createPolicy					212
createPublisher				AmSender	
createReceiver					213
createSender			201	close	213
createSubscriber			201	enableWarnings	
deleteDistributionList				getCCSID	
deleteMessage			202	getEncoding	
deletePolicy				getLastErrorStatus	
deletePublisher			202	getName	
deleteReceiver			202	open	
deleteSender			202	send	
deleteSubscriber			202	sendFile	215
enableWarnings				AmReceiver	216
getLastErrorStatus				browse	216
getName			203	clearErrorCodes	
getTraceLevel			203	close	217
getTraceLocation			203	enableWarnings	217
open			203	getDefinitionType	217
rollback			203	getLastErrorStatus	218
AmMessage			204	getName	218
addElement			205	getQueueName	218
addFilter			205	open	218
addTopic			205	receive	218
clearErrorCodes			205	receiveFile	219
deleteElement			205	setQueueName	219
deleteFilter			205	AmDistributionList	220
deleteNamedElement			206	clearErrorCodes	220
deleteTopic			206	close	220
enableWarnings			206	enableWarnings	
getCCSID			206	getLastErrorStatus	
getCorrelationId				getName	
getDataLength			206	getSender	
getDataOffset			206	getSenderCount	220
getElement				open	220
getElementCCSID				send	
getElementCount				sendFile	
getEncoding			207	AmPublisher	222

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
getName	AmErrorException
getQueueName	getClassName
open	getCompletionCode
receive	getMethodName
setQueueName	getReasonCode
subscribe	getSource
unsubscribe	20.
AmPolicy	AmWarningException
clearErrorCodes	getClassName
	getCompletionCode
enableWarnings	getMethodName
ě	getReasonCode
0	
0	getSource
setWaitTime	toString
AmBytes	
cmp	
constructors	
cpy	
dataPtr	
destructor	
length	
operators	
pad	
AmElement	
AmElement	
getName	
getValue	
getVersion	
setVersion	
toString	
AmObject	
clearErrorCodes	
getLastErrorStatus	
getName	
AmStatus	
AmStatus	
getCompletionCode	
getReasonCode	
getReasonCode2	
toString	
AmString	
cat	
cmp	
222	

Chapter 7. Using the Application Messaging Interface in C++

The Application Messaging Interface for C++ (amCpp) provides a C++ style of programming, while being consistent with the object-style interface of the Application Messaging Interface for C.

This chapter describes the following:

- "Structure of the AMI"
- "Writing applications in C++" on page 165
- "Building C++ applications" on page 175

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 492).

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

System default objects

The set of system default objects created in C is not accessible directly in C++, but the SYSTEM.DEFAULT.POLICY (constant: AMSD_POL) is used to provide default behavior when a policy is not specified. Objects with identical properties to the system default objects can be created for use in C++ using the built-in definitions (see "System provided definitions" on page 492).

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 185.

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 209) 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 481).

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 481).

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 166, 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 167, 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 167).

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 481).

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 478).

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 481).

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 = %1d\n", status.getReasonCode());
    amex.getSource()->clearErrorCodes();
}
```

The catch block works because all objects that throw the AmException in the try block are AmObjects, and so they all have getName, getLastErrorStatus and clearErrorCodes methods.

Transaction support

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

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

- · 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.

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

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 445 (AIX), page 449 (AS/400), page 455 (HP-UX), page 463 (Solaris), or page 468 (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 176
- "C++ applications on HP-UX" on page 177
- "C++ applications on Solaris" on page 178
- "C++ applications on Windows" on page 179

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 442 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 444).

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

ln -s /usr/mqm/lib/libamtCpp.a /usr/lib/libamtCpp.a

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

ln -s /usr/mqm/lib/libamtCpp r.a /usr/lib/libamtCpp r.a

You also need access to the C libraries and 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 442 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

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 442 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 453).

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 442 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 461).

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/mgm/lib/amtcmgic /usr/lib/amtcmgic

C++ applications on Windows

This section explains what you have to do to prepare and run your C++ programs on the Windows 98, Windows NT, Windows Me, and Windows 2000 operating systems. See "Language compilers" on page 442 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 8. C++ interface overview

This chapter contains an overview of the structure of the Application Messaging Interface for C++. Use it to find out what functions are available in this interface.

The C++ interface provides sets of methods for each of the classes in the following lists. The methods available for each class are listed in the following pages. Follow the page references to see the reference information for each method.

Base classes

AmSessionFactory	page 182
AmSession	page 183
AmMessage	page 185
AmSender	page 187
AmReceiver	page 188
AmDistributionList	page 189
AmPublisher	page 190
AmSubscriber	page 191
AmPolicy	page 192

Helper classes

AmBytes	page 193
AmElement	page 193
AmObject	page 193
AmStatus	page 193
AmString	page 194

Exception classes

AmException	page 195
AmErrorException	page 195
AmWarningExcpetion	page 195

AmSessionFactory

The AmSessionFactory class is used to create AmSession objects.

Constructor

Constructor for AmSessionFactory.

AmSessionFactory page 198

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 198
getLocalHost	page 198
getRepository	page 198
getTraceLevel	page 198
getTraceLocation	page 198
setLocalHost	page 199
setRepository	page 199
setTraceLevel	page 199
setTraceLocation	page 199

Create and delete session

Methods to create and delete an AmSession object.

create Sessionpage 198 page 198 deleteSession

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 203
close	page 200
getName	page 203
getTraceLevel	page 203
getTraceLocation	page 203

Create objects

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

createMessage	page 201
createSender	page 201
createReceiver	page 201
createDistributionList	page 200
createPublisher	page 201
createSubscriber	page 201
createPolicy	page 201

Delete objects

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

deleteMessage	page 202
deleteSender	page 202
deleteReceiver	page 202
deleteDistributionList	page 202
deletePublisher	page 202
deleteSubscriber	page 202
deletePolicy	page 202

Transactional processing

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

begin	page 200
commit	page 200
rollback	page 203

C++ interface overview

Error handling

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

clear Error Codespage 200 enableWarnings page 202 page 203 getLastErrorStatus

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 206
page 206
page 207
page 207
page 207
page 208
page 208
page 208
page 209
page 209

Set values

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

setCCSID	page 210
setCorrelationId	page 210
setElementCCSID	page 210
setEncoding	page 210
setFormat	page 211
setGroupStatus	page 211
setReportCode	page 211
setType	page 212

Reset values

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

reset page 209

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 206
getDataOffset	page 206
setDataOffset	page 210
readBytes	page 209

C++ interface overview

writeBytes page 212

Publish/subscribe topics

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

addTopicpage 205deleteTopicpage 206getTopicpage 209getTopicCountpage 209

Publish/subscribe filters

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

addFilterpage 205deleteFilterpage 205getFilterpage 207getFilterCountpage 207

Publish/subscribe name/value elements

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

addElementpage 205deleteElementpage 205getElementpage 207getElementCountpage 207deleteNamedElementpage 206getNamedElementpage 208getNamedElementCountpage 208

Error handling

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

clearErrorCodespage 205enableWarningspage 206getLastErrorStatuspage 208

AmSender

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

Open and close

Methods to open and close the sender service.

page 214 open close page 213

Send

Method to send a message.

send page 214

Send file

Method to send data from a file

sendFile page 215

Get values

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

getCCSID page 213 getEncoding page 213 getName page 214

Error handling

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

clearErrorCodes page 213 enableWarnings page 213 getLastErrorStatuspage 214

AmReceiver

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

Open and close

Methods to open and close the receiver service.

page 218 open close page 217

Receive and browse

Methods to receive or browse a message. receive page 218

browse page 216

Receive file

Method to receive file message data into a file.

receiveFile page 219

Get values

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

getDefinitionType page 217 getName page 218 getQueueName page 218

Set value

Method to set the queue name of the receiver service.

setQueueName page 219

Error handling

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

clearErrorCodes page 217 enableWarnings page 217 getLastErrorStatus page 218

AmDistributionList

An AmDistributionList object encapsulates a list of AmSender objects.

Open and close

Methods to open and close the distribution list service.

open page 220close page 220

Send

Method to send a message to the distribution list.

send page 221

Send file

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

sendFile page 221

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.

getNamepage 220getSenderCountpage 220getSenderpage 220

Error handling

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

clearErrorCodespage 220enableWarningspage 220getLastErrorStatuspage 220

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 223 open close page 222

Publish

Method to publish a message.

publish page 223

Get values

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

getCCSID page 222 page 222 getEncoding getName page 222

Error handling

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

clearErrorCodes page 222 enableWarnings page 222 getLastErrorStatus page 222

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 225 open close page 224

Broker messages

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

subscribe page 226 unsubscribe page 226 receive page 225

Get values

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

getCCSID page 224 getDefinitionType page 224 getEncoding page 224 getName page 225 getQueueName page 225

Set value

Method to set the queue name of the subscriber service.

setQueueName page 225

Error handling

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

clearErrorCodes page 224 enableWarnings page 224 getLastErrorStatus page 224

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 227 getWaitTime page 227 setWaitTime page 227

Error handling

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

clearErrorCodes page 227 enableWarnings page 227 page 227 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 228
destructor	page 229
operators	page 229
cmp	page 228
сру	page 229
dataPtr	page 229
length	page 229
pad	page 229

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 230
getName	page 230
getValue	page 230
getVersion	page 230
setVersion	page 230
toString	page 230

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 231
getLastErrorStatus	page 231
getName	page 231

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 232
getCompletionCode	page 232

C++ interface overview

getReasonCode	page 232
getReasonCode2	page 232
toString	page 232

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 233
destructor	page 233
operators	page 234
cat	page 233
cmp	page 233
contains	page 233
сру	page 233
length	page 234
pad	page 234
split	page 234
strip	page 234
text	page 234
truncate	page 234

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 235
get Completion Code	page 235
getMethodName	page 235
getReasonCode	page 235
getSource	page 235
toString	page 235

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 236
get Completion Code	page 236
getMethodName	page 236
getReasonCode	page 236
getSource	page 236
toString	page 236

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 237
get Completion Code	page 237
getMethodName	page 237
getReasonCode	page 237
getSource	page 237
toString	page 237

C++ interface overview

Chapter 9. C++ interface reference

In the following sections the C++ interface methods are listed by the class they refer to. Within each section the methods are listed in alphabetical order.

Base classes

Note that all of the methods in these classes can throw AmWarningException and AmErrorException (see below). However, by default, AmWarningExceptions are not raised.

AmSessionFactory	page 198
AmSession	page 200
AmMessage	page 204
AmSender	page 213
AmReceiver	page 216
AmDistributionList	page 220
AmPublisher	page 222
AmSubscriber	page 224
AmPolicy	page 227

Helper classes

AmBytes	page 228
AmElement	page 230
AmObject	page 231
AmStatus	page 232
AmString	page 233

Exception classes

AmException	page 235
AmErrorException	page 236
AmWarningException	page 237

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 471 for the location of these files.

createSession

Creates an AmSession object.

```
AmSession * createSession(char * name);
```

The name of the AmSession.

deleteSession

Deletes an AmSession object previously created using the createSession method. void deleteSession(AmSession ** pSession);

pSession

A pointer to the AmSession pointer returned by the createSession method.

getFactoryName

Returns the name of the AmSessionFactory. AmString getFactoryName();

getLocalHost

Returns the name of the local host file.

```
AmString getLocalHost();
```

getRepository

Returns the name of the repository file.

```
AmString getRepository();
```

getTraceLevel

Returns the trace level for the AmSessionFactory. int getTraceLevel();

getTraceLocation

Returns the location of the trace for the AmSessionFactory.

```
AmString getTraceLocation();
```

setLocalHost

Sets the name of the AMI local host file to be used by any AmSession created from this AmSessionFactory. (Otherwise, the default host file amthost.xml is used.)

void setLocalHost(char * fileName);

fileName

The name of the file used by the AMI as the local host file. This file must be present on the local file system or an error will be produced upon the creation of an AmSession.

setRepository

Sets the name of the AMI repository to be used by any AmSession created from this AmSessionFactory. (Otherwise, the default repository file amt.xml is used.) void setRepository(char * fileName);

fileName

Either of the following:

- The name of the file used by the AMI as the repository. This file must be present on the local file system or an error will be produced upon the creation of an AmSession.
- A reference to the repository information in LDAP URL format, when repository information is obtained from an LDAP directory.

For details about specifying an LDAP URL, see "Directory search" on page 512.

setTraceLevel

Sets the trace level for the AmSessionFactory.

void setTraceLevel(int level);

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(AMB00L warnings0n);

If set to AMB_TRUE, AmWarningExceptions will be raised for this warnings0n 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

groupStatus AMGRP_MSG_NOT_IN_GROUP

reportCode AMFB_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 209) and rebuild it each time.

Note that the following methods are only valid after a session has been opened with AmSession.open or after you have explicitly set the element CCSID with AmMessage.setElementCCSID:

addElement	page 205
deleteElement	page 205
getElement	page 207
getElementCount	page 207
delete Name d Element	page 206
getNamedElement	page 208
get Named Element Count	page 208
addTopic	page 205
deleteTopic	page 206
getTopic	page 209
getTopicCount	page 209

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

The name of the topic to be added to the AmMessage.

clearErrorCodes

Clears the error in the AmMessage object.

```
void clearErrorCodes();
```

deleteElement

Deletes the element in the AmMessage object at the specified index. Indexing is within all elements of a message, and might include topics (which are specialized elements).

```
void deleteElement(int index);
```

index

The index of the element to be deleted, starting from zero. On completion, elements with higher index values than that specified will have those values reduced by one.

getElementCount gets the number of elements in the message.

deleteFilter

Deletes a publish/subscribe filter in an AmMessage object at the specified index. Indexing is within all filters in the message.

```
void deleteFilter(int filterIndex);
```

filterIndex

The index of the filter to be deleted, starting from zero. **getFilterCount** gets the number of filters in a message.

deleteNamedElement

Deletes the element with the specified name in the AmMessage object, at the specified index. Indexing is within all elements that share the same name.

```
void deleteNamedElement(
  char * name,
  int index);
```

name The name of the element to be deleted.

index The index of the element to be deleted, starting from zero. On

completion, elements with higher index values than that specified

will have those values reduced by one.

getNamedElementCount gets the number of elements in the

message with the specified name.

deleteTopic

Deletes a publish/subscribe topic in an AmMessage object at the specified index. Indexing is within all topics in the message.

```
void deleteTopic(int index);
```

index The index of the topic to be deleted, starting from zero.

getTopicCount gets the number of topics in the message.

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void enableWarnings(AMBOOL warningsOn);
```

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_IN_GROUP
AMGF_FIRST
AMGF_LAST
```

getLastErrorStatus

Returns the AmStatus of the last error condition for this object.

```
AmStatus getLastErrorStatus();
```

getMessageId

Returns the message identifier from the AmMessage object.

```
AmBytes getMessageId();
```

getName

Returns the name of the AmMessage object.

```
AmString getName();
```

getNamedElement

Returns the element with the specified name in an AmMessage object, at the specified index. Indexing is within all elements that share the same name.

```
AmElement getNamedElement(
  char * name,
  int index);
```

name The name of the element to be returned.

index The index of the element to be returned, starting from zero.

getNamedElementCount

Returns the total number of elements with the specified name in the AmMessage object.

```
int getNamedElementCount(char * name);
```

name The name of the elements to be counted.

getReportCode

Returns the feedback code from an AmMessage of type AMMT_REPORT.

```
int getReportCode();
```

In addition to application defined values, the following values can be returned:

```
AMFB_NONE
AMFB_CODE_EXPIRATION
AMFB_CODE_COA
AMFB_CODE_COD
```

getTopic

Returns the publish/subscribe topic in the AmMessage object, at the specified index. Indexing is within all topics.

```
AmString getTopic(int index);
```

index

The index of the topic to be returned, starting from zero. **getTopicCount** gets the number of topics in the message.

getTopicCount

Returns the total number of publish/subscribe topics in the AmMessage object. int getTopicCount();

getType

Returns the message type from the AmMessage.

```
int getType();
```

The following values can be returned:

```
AMMT_REQUEST
AMMT_REPLY
AMMT_REPORT
AMMT_DATAGRAM
```

readBytes

Populates an AmByte object with data from the AmMessage, starting at the current data offset (which must be positioned before the end of the data for the read to be successful). Use **setDataOffset** to specify the data offset. **readBytes** will advance the data offset by the number of bytes read, leaving the offset immediately after the last byte read.

```
AmBytes readBytes(int dataLength);
```

dataLength

The maximum number of bytes to be read from the message data. The number of bytes returned is the minimum of dataLength and the number of bytes between the data offset and the end of the data.

reset

Resets the AmMessage object to its initial state (see page 204).

```
reset is overloaded: the options parameter is optional.
  void   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 an empty AmBytes object.

setDataOffset

Sets the data offset for reading or writing byte data.

void setDataOffset(int dataOffset);

dataOffset

The data offset to be set in the AmMessage. Set an offset of zero to read or write from the start of the data.

setElementCCSID

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

void setElementCCSID(int elementCCSID);

elementCCSID The element CCSID to be set in the AmMessage.

setEncoding

Sets the encoding of the data in the AmMessage object.

void setEncoding(int encoding);

encoding

The encoding to be used in the AmMessage. It can take one of the following values:

AMENC NATIVE AMENC NORMAL AMENC_NORMAL_FLOAT 390 AMENC_REVERSED

AMENC REVERSED FLOAT 390

AMENC UNDEFINED

setFormat

Sets the format for the AmMessage object.

```
void setFormat(char * format);
```

format

The format to be used in the AmMessage. It can take one of the following values:

AMFMT_NONE AMFMT_STRING AMFMT_RF_HEADER

If set to AMFMT_NONE, the default format for the sender will be used (if available). Specify as NULL to set the format to NULL.

setGroupStatus

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

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

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

```
void setGroupStatus(int groupStatus);
```

groupStatus

The group status to be set in the AmMessage. It can take one of the following values:

AMGRP_MSG_NOT_IN_GROUP AMGRP_FIRST_MSG_IN_GROUP AMGRP_MIDDLE_MSG_IN_GROUP AMGRP_LAST_MSG_IN_GROUP AMGRP_ONLY_MSG_IN_GROUP

setReportCode

Sets the feedback code used by the AmMessage object. This is meaningful only for a message of type AMMT_REPORT.

void setReportCode(int reportCode);

reportCode

The feedback (or report code) value set in the AmMessage.

In addition to application defined values, the following values can be set:

AMFB_NONE AMFB_CODE_EXPIRATION AMFB_CODE_COA AMFB_CODE_COD

211

C++ AmMessage

setType

Sets the message type used by the AmMessage object. If a response message is requested with a publish, subscribe, or unsubscribe request, the specified value is ignored and message type AMMT_REQUEST is used. If the value specified is AMMT_DATAGRAM, this is overridden when requesting or sending a response message (by AMMT_REQUEST and AMMT_RESPONSE, respectively).

void setType(int type);

type

The message type to be set in the AmMessage. It can take one of the following values:

AMMT DATAGRAM AMMT REQUEST AMMT REPLY AMMT REPORT

writeBytes

Writes a byte array into the AmMessage object, starting at the current data offset. If the data offset is not at the end of the data, existing data is overwritten. Use setDataOffset to specify the data offset. writeBytes will advance the data offset by the number of bytes written, leaving it immediately after the last byte written.

void writeBytes(AmBytes &data);

data

An AmBytes object containing the data to be written to the AmMessage.

AmSender

An AmSender object encapsulates 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.

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 219 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 NEXT)
AMBRW LOCK NEXT
                       (AMBRW LOCK + AMBRW NEXT)
AMBRW LOCK FIRST
                       (AMBRW LOCK + AMBRW FIRST)
AMBRW LOCK CURRENT
                       (AMBRW LOCK + AMBRW CURRENT)
AMBRW UNLOCK
```

AMBRW RECEIVE CURRENT is equivalent to **AmReceiver.receive** for the message under the browse cursor.

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

responseService

The AmSender to be used for sending any response to the browsed message. If omitted, no response can be sent.

Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.

selectionMessage

A message object which contains the Correlation ID used to selectively browse a message from the AmReceiver. If omitted, the first available message is browsed. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be be reset (see AmMessage.setCorrelationId on page 210) 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 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 210) 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 210) 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

getSenderCount

Returns the number of AmSender services in the AmDistributionList object.

```
int getSenderCount();
```

open

Opens an AmDistributionList object for each of the destinations in the distribution list. **open** is overloaded: the policy parameter is optional.

```
void open(AmPolicy &policy);
```

policy

The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

send

Sends a message to each AmSender defined in the AmDistributionList object. **send** is overloaded: the sendMessage parameter is required, but the others are optional.

void send(
AmMessage &sendMessage,
AmReceiver &responseService,
AmPolicy &policy);

sendMessage The message object containing the data to be sent.

responseService

The AmReceiver to be used for receiving any response to the sent message. If omitted, no response can be received.

policy The policy to be used. If omitted, the system default policy

(constant: AMSD_POL) is used.

sendFile

Sends data from a file to each AmSender defined in the AmDistributionList object. The sendMessage and fileName parameters are required to send data from a file, but the policy is optional. The file data can be received as normal message data by a target application using AmReceiver.receive, or used to reconstruct the file with AmReceiver.receiveFile.

void sendFile(
 AmMessage &sendMessage,
 char* fileName,
 AmPolicy &policy);

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

specify the Correlation ID, for example. The message must not

include any elements or data.

fileName The name of the file to be sent (input). This can include a directory

prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel

with the message for use with the receive file method (see

"receiveFile" on page 219 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(AMB00L warnings0n);
```

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();
```

open

Opens an AmPublisher service. **open** is overloaded: the policy parameter is optional.

```
void open(AmPolicy &policy);
```

policy

The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

publish

Publishes a message using the AmPublisher. **publish** is overloaded: the pubMessage parameter is required, but the others are optional.

void publish(

AmMessage &pubMessage,
AmReceiver &responseService,
AmPolicy &policy);

pubMessage The message object that contains the data to be published.

responseService

The AmReceiver to which the response to this publish request should be sent. Omit it if no response is required. This parameter is mandatory if the policy specifies implicit registration of the publisher.

policy

The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmSubscriber

An AmSubscriber object encapsulates both an AmSender and an AmReceiver. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

clearErrorCodes

```
Clears the error codes in the AmSubscriber.
 void clearErrorCodes();
```

close

Closes the AmSubscriber. close is overloaded: the policy parameter is optional. void close(AmPolicy &policy);

policy

The policy to be used. If omitted, the system default policy (constant: AMSD POL) is used.

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using getLastErrorStatus, even if AmWarningExceptions are disabled.

```
void enableWarnings(AMB00L warnings0n);
```

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();
```

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 204).

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 210) 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.

C++ AmSubscriber

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 204) 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 204) before sending the message.

unsubscribe is overloaded: the unsubMessage parameter is required, but the others are optional.

```
void unsubscribe(
  AmMessage &unsubMessage,
 AmReceiver &responseService,
  AmPolicy
             &policy);
```

unsubMessage The message object that contains the topics to which the

unsubscribe request applies.

responseService

The AmReceiver to which the response to this unsubscribe request should be sent. Omit it if no response is required.

policy The policy to be used. If omitted, the system default policy

(constant: AMSD_POL) is used.

AmPolicy

An AmPolicy object encapsulates details of how the AMI processes the message (for instance, the priority and persistence of the message, how errors are handled, and whether transactional processing is used).

clearErrorCodes

Clears the error codes in the AmPolicy. void clearErrorCodes();

enableWarnings

Enables AmWarningExceptions; the default behavior for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using getLastErrorStatus, even if AmWarningExceptions are disabled.

void enableWarnings(AMBOOL warningsOn);

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.

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.

The single byte being copied.

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);
```

version

The version of the AmElement that is set. It can take the value AMELEM_VERSION_1 or AMELEM_CURRENT_VERSION.

toString

```
Returns a AmString representation of the AmElement.
```

```
AmString toString();
```

AmObject

AmObject is a virtual class. The following classes inherit from the AmObject class:

AmSession

AmMessage

AmSender

AmDistributionList

AmReceiver

AmPublisher

AmSubscriber

AmPolicy

This allows application programmers to use generic error handling routines.

clearErrorCodes

Clears the error codes in the AmObject.

void clearErrorCodes();

getLastErrorStatus

Returns the AmStatus of the last error condition.

AmStatus getLastErrorStatus();

getName

Returns the name of the AmObject.

AmString getName();

AmStatus

An **AmStatus** object encapsulates the error status of other AmObjects.

AmStatus

```
Constructor for an AmStatus object.
 AmStatus();
```

getCompletionCode

```
Returns the completion code from the AmStatus object.
 int getCompletionCode();
```

getReasonCode

```
Returns the reason code from the AmStatus object.
 int getReasonCode();
```

getReasonCode2

Returns the secondary reason code from the AmStatus object. (This code is specific to the underlying transport used by the AMI). For 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;
```

amString A reference to the AmString object being compared.

stringData The NULL terminated string being compared.

constructors

Constructors for an AmString object.

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

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

AmString.

stringData A NULL terminated string, from which the AmString is

constructed.

contains

Method to indicate whether a specified character is contained within the AmString.

AMBOOL contains(const AMBYTE character) const;

character The character being used for the search.

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);
```

amString A reference to an AmString object being copied.

stringData The NULL terminated string being copied into the AmString.

destructor

Destructor for an AmString object.

```
~AmString();
```

C++ 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);

A reference to an AmString object to contain the latter half of the newString 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();

C++ AmWarningException

Part 4. The COBOL interface

Chapter 10. Using the Application Messaging	AMHRCRQ (receive request)	276
Interface in COBOL	AMHSNFL (send file)	
Structure of the AMI	AMHSNMS (send message)	279
Using the repository	AMHSNRQ (send request)	280
System default objects 244	AMHSNRS (send response)	
Writing applications in COBOL 246	AMHSB (subscribe)	
Opening and closing a session 246	AMHTERM (terminate)	
Sending messages 246	AMHUN (unsubscribe)	
Using the message object 247		
Sample programs	Chapter 12. COBOL object interface overview 2	85
Receiving messages 248	Session interface functions	
Using the message object 249	Session management	
Sample programs 249	Create objects	
Request/response messaging	Get object handles	
Request	Delete objects	287
Response	Transactional processing	
Sample programs	Error handling	
File transfer	Message interface functions	
Publish/subscribe messaging 251	Get values	
Publish	Set values	
Subscribe	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 2	
Transaction support	Error handling	
Sending group messages	Sender interface functions	
Other considerations	Open and close	
Multithreading	Send	
Using MQSeries with the AMI	Get values	
Field limits	Error handling	
Building COBOL applications	Receiver interface functions	.)0)01
COBOL applications on OS/390	Open and close	
AMI Copybooks	Receive and browse	
Preparing COBOL programs on OS/390	Get values	
Running COBOL programs on OS/390 258	Set values	
Rumming Coboli programs on Co7 570 200	Error handling	
Chapter 11. The COBOL high-level interface 259	Distribution list interface functions	
Overview of the COBOL high-level interface 260	Open and close	
Initialize and terminate	Send	
	Get values	
Sending messages		
Receiving messages	Error handling	
File transfer	Publisher interface functions	
Publish/subscribe	Open and close	
Transaction support	Publish	
Reference information for the COBOL high-level	Get values	
interface	Error handling	
AMHBACK (backout)	Subscriber interface functions	
AMHBEGIN (begin)	Open and close	
AMHBRMS (browse message)	Broker messages	
AMHCMIT (commit)	Get values	
AMHINIT (initialize)	Set value	
AMHPB (publish)	Error handling	
AMHRCFL (receive file)	Policy interface functions	
AMHRCMS (receive message)	Get values	
AMHRCPB (receive publication)	Set value	95

Error handling	. 295	AMMSGTRC (get report code)		. 325
High-level functions		AMMSGTTO (get topic)		. 325
		AMMSGTTC (get topic count)		
Chapter 13. COBOL object interface reference	299	AMMSGTTY (get type)		. 326
Session interface functions	. 300	AMMSREBY (read bytes)		
AMSEBG (begin)		AMMSRS (reset)		. 327
AMSECLEC (clear error codes)		AMMSSTCC (set CCSID)		. 328
AMSECL (close)		AMMSSTCI (set correl ID)		. 328
AMSECM (commit)	. 301	AMMSSELC (set element CCSID)		. 328
AMSECR (create)		AMMSSTDO (set data offset)		
AMSECRDL (create distribution list)		AMMSSTEN (set encoding)		
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)		334
		AMSNOP (open)		
AMSEDLPB (delete publisher)		AMSNSN (send)		
		AMSNSNFL (send file)		
AMSEDLSN (delete sender)		Usage notes	• •	. 336
AMSECUEL (act distribution list bondle)		Receiver interface functions		. 338
AMSEGTIF (get distribution list handle)		AMRCBR (browse)		
AMSECTIMS (get last error codes)				
AMSEGHMS (get message handle)		Usage notes		
AMSEGHPO (get policy handle)		AMRCBRSE (browse selection message)		
AMSEGHPB (get publisher handle)		Usage notes		
AMSEGHRC (get receiver handle)		AMRCCLEC (clear error codes)		
AMSEGHSN (get sender handle)		AMRCCL (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)		. 344
AMMSADEL (add element)		AMRCOP (open)		. 344
AMMSADFI (add filter)		AMRCRC (receive)		
AMMSADTO (add topic)	. 315	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)	. 317	AMDLCL (close)		
AMMSGELC (get element CCSID)		AMDLGTLE (get last error)		
AMMSGTCC (get CCSID)	. 317	AMDLGTNA (get name)		
AMMSGTCI (get correl ID)		AMDLGTSC (get sender count)		
AMMSGTDL (get data length)	. 318	AMDLGTSH (get sender handle)		
AMMSGTDO (get data offset)		AMDLOP (open)		
AMMSGTEL (get element)		AMDLSN (send)		. 351
AMMSGTEC (get element count)	. 319	AMDLSNFL (send file)		. 351
AMMSGTEN (get encoding)		Usage notes		
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				357
AMSBCLEC (clear error codes)				357
AMSBCL (close)				357
AMSBGTCC (get CCSID)				358
AMSBGTDT (get definition type)				358
AMSBGTEN (get encoding) .				359
AMSBGTLE (get last error)				359
AMSBGTNA (get name)				360
AMSBGTQN (get queue name)				360
AMSBOP (open)				361
AMSBRC (receive)				361
AMSBSTQN (set queue name).				362
AMSBSB (subscribe)				362
AMSBUN (unsubscribe)				363
Policy interface functions				364
AMPOCLEC (clear error codes)				364
AMPOGTLE (get last error) .				364
AMPOGTNA (get name)				365
AMPOGTWT (get wait time) .				365
AMPOSTWT (set wait time) .				366

Chapter 10. Using the Application Messaging Interface in COBOL

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

- 1. A high-level procedural interface that provides the function needed by the majority of users.
- 2. A lower-level, object-style interface, that provides additional function for experienced MQSeries users.

This chapter describes the following:

- "Structure of the AMI"
- "Writing applications in COBOL" on page 246
- "Building COBOL applications" on page 257

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 11. The COBOL high-level interface" on page 259. An overview of the object interface is given in "Chapter 12. COBOL object interface overview" on page 285, with reference information in "Chapter 13. COBOL object interface reference" on page 299.

Using the repository

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

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

System default objects

Table 3. System default objects

Default object	Constant or handle (if applicable)			
SYSTEM.DEFAULT.POLICY	AMSD-POL AMSD-POL-HANDLE			
SYSTEM.DEFAULT.SYNCPOINT.POLICY	AMSD-SYNC-POINT-POL AMSD-SYNC-POINT-POL-HANDLE			
SYSTEM.DEFAULT.SENDER	AMSD-SND			
SYSTEM.DEFAULT.RESPONSE.SENDER	AMSD-RSP-SND AMSD-RSP-SND-HANDLE			
SYSTEM.DEFAULT.RECEIVER	AMSD-RCV AMSD-RCV-HANDLE			
SYSTEM.DEFAULT.PUBLISHER	AMSD-PUB AMSD-PUB-SND			
SYSTEM.DEFAULT.SUBSCRIBER	AMSD-SUB AMSD-SUB-SND			
SYSTEM.DEFAULT.SEND.MESSAGE	AMSD-SND-MSG AMSD-SND-MSG-HANDLE			
SYSTEM.DEFAULT.RECEIVE.MESSAGE	AMSD-RCV-MSG AMSD-RCV-MSG-HANDLE			

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

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 and structures" on page 561). Note that the first parameter on a call must be a real handle; you cannot use a synonym in this case (that is why handles are not provided for all the default objects).

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 296).

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 268):

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 283):

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 279). In the simplest case, all you need to specify is the session handle returned by AMHINIT, the message data, and the message length. Other parameters can be specified using the constants that represent the default message, sender service, and policy objects.

Sending a message using all the defaults

CALL 'AMHSNMS' USING HSESSION, AMSD-SND, AMSD-POL, DATALEN, DATA, AMSD-SND-MSG, COMPCODE, REASON.

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

Sending a message using a specified sender service

CALL 'AMHSNMS' USING HSESSION, SENDER-NAME, AMSD-POL, DATALEN, DATA, AMSD-SND-MSG, COMPCODE, REASON.

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

Sending a message using a specified policy

CALL 'AMHSNMS' USING HSESSION, AMSD-SND, POLICY-NAME, DATALEN, DATA, AMSD-SND-MSG, COMPCODE, REASON.

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

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

To send a message to a distribution list, specify its name (such as DISTLIST-NAME) as the sender service:

Sending a message to a distribution list

CALL 'AMHSNMS' USING HSESSION, DISTLIST-NAME, AMSD-POL, DATALEN, DATA, AMSD-SND-MSG, COMPCODE, REASON.

Using the message object

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

Specifying a message object

CALL 'AMHSNMS' USING HSESSION, AMSD-SND, AMSD-POL, DATALEN, DATA, SEND-MSG, COMPCODE, REASON.

The AMI creates a message object of the name specified (SEND-MSG), if one doesn't already exist. (In this example the defaults for the sender name and policy name are used.) You can then use object interface functions to get the required attributes, such as the *MessageID*, from the message object:

Getting an attribute from a message object

CALL 'AMSEGHMS' USING HSESSION, SEND-MSG, HMSG, COMPCODE, REASON.

CALL 'AMMSGTMI' USING HMSG, BUFFLEN, MSGIDLEN, MSGID, COMPCODE, REASON.

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

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

Setting an attribute in a message object

```
CALL 'AMSECRMS' USING HSESSION, SEND-MSG, HMSG, COMPCODE, REASON.
CALL 'AMMSSTFO' USING HMSG, FORMATLEN, FORMAT, COMPCODE, REASON.
```

Then you can send the message as before, making sure to specify the same message object name (SEND-MSG) in the AMHSNMS call.

Look at "Message interface functions" on page 288 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 327) 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 484).

Receiving messages

Use the AMHRCMS high-level function (page 272) 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 Correlld 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 484).

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 280) 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 276) 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 248.

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 281):

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 248. 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 484).

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 478).

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

Use the AMHPB high-level function (page 269) 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 282) 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 284). 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 274). 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 248.

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 484).

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.
                               PIC X(16) VALUE 'MQPSCommand'.
   10 COMMAND-NAME
                               PIC X(16) VALUE 'ReqUpdate'
   10 COMMAND-VALUE
   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

```
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 442 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:

```
EXEC PGM=HEWL....
//LKED
//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 529.

Chapter 11. The COBOL high-level interface

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

This chapter contains:

- "Overview of the COBOL high-level interface" on page 260
- "Reference information for the COBOL high-level interface" on page 262

Overview of the COBOL high-level interface

The high-level functions are listed below. Follow the page references to see the detailed descriptions of each function.

Initialize and terminate

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

AMHINIT (initialize) page 268 AMHTERM (terminate) page 283

Sending messages

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

AMHSNMS (send message) page 279
AMHSNRQ (send request) page 280
AMHSNRS (send response) page 281

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 272

AMHRCRQ (receive request) page 276

AMHBRMS (browse message)

page 265

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 278
AMHRCFL (receive file) page 270

Publish/subscribe

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

AMHPB (publish) page 269
AMHSB (subscribe) page 282
AMHUN (unsubscribe) page 284

AMHRCPB (receive publication)

page 274

Transaction support

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

AMHBEGIN (begin) page 264

COBOL high-level interface overview

page 267 AMHCMIT (commit) page 263 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 and LDAP error codes" on page 537).

Object names can be up to AMLEN-MAX-NAME-LENGTH characters, and are terminated by a space or by a low value (a single byte zero). If a space or low value is not found, the name will be truncated at AMLEN-MAX-NAME-LENGTH.

If an object name is specified as a space or low value, the relevant system default name will be used.

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

AMHBACK (backout)

Function to back out a unit of work.

CALL 'AMHBACK' USING HSESSION, POLICY, COMPCODE, REASON.

Declare the parameters as follows:

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

HSESSION The session handle returned by AMHINIT (input).

POLICY The name of a policy (input). If specified as a space or low value,

the system default policy name (constant: AMSD-POL) is used.

COMPCODE Completion code (output).

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. PIC X(n). 01 POLICY 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 313). 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.

Declare the parameters as follows:

01 SESSNAME 01 POLICY 01 HSESSION 01 COMPCODE 01 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.	
HSESSION The session handle (output).	
COMPCODE 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 S	9(9) BINARY.
01	PUBLISHER	PIC X	(n).
01	POLICY	PIC X	(n).
01	RESPNAME	PIC X	(n).
01	TOPICLEN	PIC S	9(9) BINARY.
01	TOPIC	PIC X	(n).
01	DATALEN	PIC S	9(9) BINARY.
01	DATA	PIC X	(n).
01	MSGNAME	PIC X	(n).
01	COMPCODE	PIC S	9(9) BINARY.
01	REASON	PIC S	9(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 313).

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 313). The message object is reset implicitly before the receive takes place.

270

COBOL high-level interface

COMPCODE Completion code (output).

REASON Reason code (output).

Usage notes

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

AMHRCMS (receive message)

Function to receive a message.

CALL 'AMHRCMS' USING HSESSION, RECEIVER, POLICY, SELMSGNAME, BUFFLEN, DATALEN, DATA, RCVMSGNAME, COMPCODE, REASON.

Declare the parameters as follows:

01	HSESSION	PIC	S9(9)	BINARY.	
01	RECEIVER	PIC	X(n).		
01	POLICY	PIC	X(n).		
01	SELMSGNAME	PIC	X(n).		
01	BUFFLEN	PIC	S9(9)	BINARY.	
01	DATALEN	PIC	S9(9)	BINARY.	
01	DATA	PIC	X(n).		
01	RCVMSGNAME	PIC	X(n).		
01	COMPCODE	PIC	S9(9)	BINARY.	
01	REASON	PIC	S9(9)	BINARY.	

HSESSION The session handle returned by AMHINIT (input).

RECEIVER The name of a receiver service (input). If specified as a space or

low value, the system default receiver name (constant:

AMSD-RCV) is used.

POLICY The name of a policy (input). If specified as a space or low value,

the system default policy name (constant: AMSD-POL) is used.

SELMSGNAME Optional selection message object used to specify information (such

as a *CorrelId*) needed to select the required message (input).

BUFFLEN The length in bytes of a buffer in which the data is returned

(input). Can be specified as -1.

DATALEN The length of the message data, in bytes (output). Can be specified

as -1 (input).

DATA The received message data (output).

RCVMSGNAME The name of the message object for the received message (output).

> If specified as a space or low value, the system default receive message name (constant: AMSD-RCV-MSG) is used. Properties, and message data if not returned in the DATA parameter, can be extracted from the message object using the object interface (see "Message interface functions" on page 313). 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.

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 313).

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 313).

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).

The received message data (output). **DATA**

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 313). 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

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

To return the data in the message object (RCVMSGNAME), rather than the application message buffer, set BUFFLEN to zero and set both DATA and DATALEN as $non_NULL (not -1).$

To return data in an application message buffer:

- set DATA as the address of the buffer (that is, non_NULL, not -1)
- set BUFFLEN to the length of the buffer

If the value of BUFFLEN is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a non_NULL value for DATALEN (that is, not −1).

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

- set DATA to NULL (-1)
- set BUFFLEN to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

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

To remove the message from the queue and discard it:

- set DATA or DATALEN to a non_NULL value (that is, not −1)
- set BUFFLEN to zero
- ensure that Accept Truncated Message in the policy receive attributes is selected

The message will be discarded with an AMRC_MSG_TRUNCATED warning.

If AMRC_RECEIVE_BUFF_LEN_ERR is returned, the message length value is returned in DATALEN (if it is non_NULL, that is, not -1), even though the completion code is MQCC_FAILED.

Note that if DATA is NULL (-1) and BUFFLEN is not zero, there is always an AMRC_RECEIVE_BUFF_LEN_ERR error.

AMHSNFL (send file)

Function to send data from a file.

```
CALL 'AMHSNFL' USING HSESSION, SENDERNAME, POLICYNAME, OPTIONS, DIRNAMELEN, DIRNAME, FILENAMELEN, FILENAME, SNDMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

01	HSESSION	PIC	S9(9)	BINARY.
01	SENDERNAME	PIC	X(n).	
01	POLICYNAME	PIC	X(n).	
01	OPTIONS	PIC	S9(9)	BINARY.
01	DIRNAMELEN	PIC	S9(9)	BINARY.
01	DIRNAME	PIC	X(n).	
01	FILENAMELEN	PIC	S9(9)	BINARY.
01	FILENAME	PIC	X(n).	
01	SNDMSGNAME	PIC	X(n).	
01	COMPCODE	PIC	S9(9)	BINARY.
01	REASON	PIC	S9(9)	BINARY.

HSESSION The session handle returned by AMHINIT (input).

SENDERNAME The name of a sender service (input). If specified as a space or low

value, the system default sender name (constant: AMSD-SND) is

used.

POLICYNAME The name of a policy (input). If specified as a space or low value,

the system default policy name (constant: AMSD-POL) is used.

OPTIONS Reserved, must be specified as zero.

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

DIRNAME Reserved.

FILENAMELEN The length of the file name in bytes (input).

FILENAME The name of the file to be sent (input). This can include a directory

prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with a receive file call (see "AMHRCFL (receive file)" on page 270 for more details). Note that the file name cent will exactly match the supplied file name; it will not be

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 313). If SNDMSGNAME is specified as a space or low value, the system default

send message name (constant: AMSD-SND-MSG) is used.

0

COMPCODE Completion code (output).

REASON Reason code (output).

Usage notes

The message object is implicitly reset by this call.

The system default object is used when you set SNDMSGNAME as a space or low value.

AMHSNMS (send message)

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

CALL 'AMHSNMS' USING HSESSION, SENDER, POLICY, DATALEN, DATA, SNDMSGNAME, COMPCODE, REASON.

Declare the parameters as follows:

```
01 HSESSION
                PIC S9(9) BINARY.
01 SENDER
                PIC X(n).
01 POLICY
                PIC X(n).
01 DATALEN
              PIC S9(9) BINARY.
01 DATA
               PIC X(n).
01 SNDMSGNAME
                PIC X(n).
01 COMPCODE
                PIC S9(9) BINARY.
01 REASON
                PIC S9(9) BINARY.
```

HSESSION The session handle returned by AMHINIT (input).

SENDER The name of a sender service (input). If specified as a space or low

value, the system default sender name (constant: AMSD-SND) is

used.

POLICY The name of a policy (input). If specified as a space or low value,

the system default policy name (constant: AMSD-POL) is used.

DATALEN The length of the message data in bytes (input). A value of zero

indicates that any message data has been added to the message object (SNDMSGNAME) using the object interface (see "Message

interface functions" on page 313).

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 313).

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 313).

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 313).

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	PIC	X(n).	
01	RESPNAME	PIC	X(n).	
01	TOPICLEN	PIC	S9(9)	BINARY.
01	TOPIC	PIC	X(n).	
01	FILTERLEN	PIC	S9(9)	BINARY.
01	FILTER	PIC	X(n).	
01	UNSUBMSGNAME	PIC	X(n).	
01	COMPCODE	PIC	S9(9)	BINARY.
01	REASON	PIC	S9(9)	BINARY.

HSESSION The session handle returned by AMHINIT (input).

SUBSCRIBER The name of a subscriber service (input). If specified as a space or

low value, the system default subscriber name (constant:

AMSD-SUB) is used.

POLICY The name of a policy (input). If specified as a space or low value,

the system default policy name (constant: AMSD-POL) is used.

RESPNAME The name of the receiver service to which the response to this

unsubscribe request will be sent (input).

TOPICLEN The length of the topic, in bytes (input).

TOPIC The topic that identifies the subscription which is to be removed

(input). Multiple topics can be specified in the message object (UNSUBMSGNAME) using the object interface (see "Message interface

functions" on page 313).

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 12. COBOL object interface overview

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

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

page 286
page 288
page 290
page 291
page 292
page 293
page 294
page 295

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 296.

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 302
AMSEOP (open)	page 312
AMSECL (close)	page 301
AMSEDL (delete)	page 306

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 303

AMSECRSN (create sender) page 305

AMSECRRC (create receiver) page 304

AMSECRDL (create distribution list)

page 302

AMSECRPB (create publisher)

page 304

AMSECRSB (create subscriber)

page 305

AMSECRPO (create policy) page 303

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 309

AMSEGHSN (get sender handle)

page 311

AMSEGHRC (get receiver handle)

page 310

AMSEGHDL (get distribution list handle)

page 308

AMSEGHPB (get publisher handle)

page 310

AMSEGHSB (get subscriber handle)

page 311

AMSEGHPO (get policy handle)

page 310

COBOL object interface overview

Delete objects

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

AMSEDLMS (delete message)

page 306

AMSEDLSN (delete sender) page 308

AMSEDLRC (delete receiver) page 307

AMSEDLDL (delete distribution list)

page 306

AMSEDLPB (delete publisher)

page 307

AMSEDLSB (delete subscriber)

page 308

AMSEDLPO (delete policy) page 307

Transactional processing

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

AMSEBG (begin) page 300 AMSECM (commit) page 301

AMSERB (rollback) page 312

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 300

AMSEGTLE (get last error codes)

page 309

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 317

AMMSGTCI (get correl ID) page 318

AMMSGELC (get element CCSID)

page 317

AMMSGTEN (get encoding) page 320

AMMSGTFO (get format) page 321

AMMSGTGS (get group status)

page 322

AMMSGTMI (get message ID)

page 323

AMMSGTNA (get name) page 323

AMMSGTRC (get report code)

page 325

AMMSGTTY (get type) page 326

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 328

AMMSSTCI (set correl ID) page 328

AMMSSELC (set element CCSID)

page 328

AMMSSTEN (set encoding) page 329

AMMSSTFO (set format) page 330

AMMSSTGS (set group status)

page 330

Reset values

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

AMMSRS (reset) page 327

Read and write data

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

AMMSGTDL (get data length)

page 318

COBOL object interface overview

AMMSGTDO (get data offset)

page 318

AMMSSTDO (set data offset) page 329

AMMSREBY (read bytes) page 327

AMMSWRBY (write bytes) page 331

Publish/subscribe topics

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

AMMSADTO (add topic) page 315

AMMSDETO (delete topic) page 317

AMMSGTTO (get topic) page 325

AMMSGTTC (get topic count)

page 326

Publish/subscribe filters

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

AMMSADFI (add filter) page 314

AMMSDEFI (delete filter) page 316

AMMSGTFI (get filter) page 321

AMMSGTFC(get filter count) page 320

Publish/subscribe name/value elements

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

AMMSADEL (add element) page 314

AMMSDEEL (delete element) page 315

AMMSGTEL (get element) page 319

AMMSGTEC (get element count)

page 319

AMMSDENE (delete named element)

page 316

AMMSGTNE (get named element)

page 324

AMMSGTNC (get named element count)

page 324

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 315

AMMSGTLE (get last error) page 322

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 335 AMSNCL (close) page 333

Send

Function to send a message.

AMSNSN (send) page 335 AMSNSNFL(send file) page 336

Get values

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

AMSNGTCC (get CCSID) page 333
AMSNGTEN (get encoding) page 333
AMSNGTNA (get name) page 334

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 332

AMSNGTLE (get last error) page 334

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 344 AMRCCL (close) page 342

Receive and browse

Functions to receive or browse a message.

AMRCRC (receive) page 345

AMRCRCFL (receive file) page 346

AMRCBR (browse) page 338

AMRCBRSE (browse selection message) page 340

Get values

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

AMRCGTDT (get definition type)

page 342

AMRCGTNA (get name) page 343

AMRCGTQN (get queue name)

page 344

Set values

Function to set the queue name of the receiver service.

AMRCSTQN (set queue name)

page 347

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 341

AMRCGTLE (get last error) page 343

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 350 AMDLCL (close) page 348

Send

Function to send a message to the distribution list.

AMDLSN (send) page 351 AMDLSNFL (send file) page 351

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 349

AMDLGTSC (get sender count)

page 349

AMDLGTSH (get sender handle)

page 350

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 348

AMDLGTLE (get last error) page 348

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 355 AMPBCL (close) page 353

Publish

Function to publish a message.

AMPBPB (publish) page 356

Get values

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

AMPBGTCC (get CCSID) page 353 AMPBGTEN (get encoding) page 354 AMPBGTNA (get name) page 355

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 353

AMPBGTLE (get last error) page 354

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 361 AMSBCL (close) page 357

Broker messages

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

AMSBSB (subscribe) page 362 AMSBUN (unsubscribe) page 363 AMSBRC (receive) page 361

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 358

AMSBGTDT (get definition type)

page 358

AMSBGTEN (get encoding) page 359

AMSBGTNA (get name) page 360

AMSBGTQN (get queue name)

page 360

Set value

Function to set the queue name of the subscriber service.

AMSBSTQN (set queue name)

page 362

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 357

AMSBGTLE (get last error) page 359

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 365 AMPOGTWT (get wait time) page 365

Set value

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

AMPOSTWT (set wait time) page 366

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 364

AMPOGTLE (get last error) page 364

High-level functions

Each high-level function described in "Chapter 11. The COBOL high-level interface" on page 259 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 13. COBOL object interface reference

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

Session	page 300
Message	page 313
Sender	page 332
Receiver	page 338
Distribution list	page 348
Publisher	page 353
Subscriber	page 357
Policy	page 364

Within each section the functions are listed in alphabetical order.

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

AMCC-OK Function completed successfully
AMCC-WARNING Function completed with a warning
AMCC-FAILED An error occurred during processing

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

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

Session interface functions

A session object provides the scope for a unit of work and creates and manages all other objects, including at least one connection object. Each (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.
01 REASON PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-

HANDLE, the system default policy (constant:

AMSD-POL-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMSECLEC (clear error codes)

Clears the error codes in the session object.

```
CALL 'AMSECLEC' USING HSESS, COMPCODE, REASON.
```

Declare the parameters as follows:

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

HSESS The session handle returned by AMSECR (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSECL (close)

Closes the session object and all open objects owned by the session, and disconnects from the underlying message transport (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).
                PIC S9(9) BINARY.
01 HSESS
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.
                 PIC S9(9) BINARY.
01 REASON
```

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 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).

REASON Reason code (output).

AMSEDLMS (delete message)

Deletes a message object.

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

Declare the parameters as follows:

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

HSESS The session handle returned by AMSECR (input).

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.
                 PIC S9(9) BINARY.
01 COMPCODE
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 253 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 327) 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 314

AMMSDEEL (delete element) page 315

AMMSGTEL (get element) page 319

AMMSGTEC (get element count)

page 319

AMMSDENE (delete named element)

page 316

AMMSGTNE (get named element)

page 324

AMMSGTNC (get named element count)

page 324

AMMSADTO (add topic) page 315

AMMSDETO (delete topic) page 317

AMMSGTTO (get topic) page 325

AMMSGTTC (get topic count)

page 326

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

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

HMSG The message handle returned by AMSECRMS (input).

AMELEM An AMELEM element structure, which specifies the element to be

added (input). It will not replace an existing element with the same

name.

OPTIONS Reserved, must be set to zero (input).

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

AMMSADFI (add filter)

Adds a filter to a subscribe or unsubscribe request message.

CALL 'AMMSADFI' USING HMSG, FILTERLEN, TOPIC, COMPCODE, REASON.

Declare the parameters as follows:

```
01 HMSG
                 PIC S9(9) BINARY.
01 FILTERLEN
                 PIC S9(9) BINARY,
01 FILTER
                 PIC X(n),
01 COMPCODE
                 PIC S9(9) BINARY.
01 REASON
                 PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

FILTERLEN The length in bytes of the filter (input). A value of

AMLEN-NULL-TERM specifies that the string is null terminated.

FILTER The filter to be added (input). The syntax of the filter string is

described in the 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	PIC	S9(9)	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).

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:

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

HMSG The message handle returned by AMSECRMS (input).

FILTERINDEX The index of the required filter in the message, starting from zero

(input). AMMSGTFI gets the number of filters in the message.

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

Reason code (output).

AMMSDENE (delete named element)

Deletes a named element from a message (such as a publish/subscribe message), at the specified index. Indexing is within all elements that share the same name.

CALL 'AMMSDENE' USING HMSG, NAMEINDEX, NAMELEN, NAME, COMPCODE, REASON.

Declare the parameters as follows:

```
PIC S9(9) BINARY.
01 HMSG
01 NAMEINDEX
                PIC S9(9) BINARY.
01 NAMELEN
                PIC S9(9) BINARY.
01
   NAME
                PIC X(n).
                PIC S9(9) BINARY.
   COMPCODE
01
                PIC S9(9) BINARY.
01
   REASON
```

HMSG The message handle returned by AMSECRMS (input).

NAMEINDEX The index of the required named element in the message (input).

> Specifying an index of zero deletes the *first* element with the specified name. On completion, elements with higher NAMEINDEX values than that specified will have their index value reduced by

one.

Use AMMSGTNC to get the number of elements in the message

with the specified name.

NAMELEN The length of the element name, in bytes (input). NAME The name of the element to be deleted (input).

COMPCODE Completion code (output).

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).

AMMSGTCI (get correl ID)

Gets the correlation identifier of the message.

```
CALL 'AMMSGTCI' USING HMSG, BUFFLEN, CORRELIDLEN, CORRELID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG PIC S9(9) BINARY.
01 BUFFLEN PIC S9(9) BINARY.
01 CORRELIDLEN PIC S9(9) BINARY.
01 CORRELID PIC X(n).
01 COMPCODE PIC S9(9) BINARY.
01 REASON PIC S9(9) BINARY.
```

The message handle returned by AMSECRMS (input).

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

returned (input).

CORRELIDLEN The length of the correlation identifier, in bytes (output).

CORRELID The correlation identifier (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTDL (get data length)

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

```
CALL 'AMMSGTDL' USING HMSG, LENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

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

The message handle returned by AMSECRMS (input).

LENGTH The length of the message data, in bytes (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTDO (get data offset)

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

```
CALL 'AMMSGTDO' USING HMSG, OFFSET, COMPCODE, REASON.
```

Declare the parameters as follows:

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

HMSG The message handle returned by AMSECRMS (input).

OFFSET The byte offset in the message data (output).

COMPCODE Completion code (output).

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).

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.

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).

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.
```

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-GROU 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).

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. PIC S9(9) BINARY. 01 REASON

HMSG The message handle returned by AMSECRMS (input).

NAMEINDEX The index of the required named element in the message (input).

Specifying an index of zero returns the first element with the

specified name.

Use AMMSGTNC to get the number of elements in the message

with the specified name.

NAMELEN The length of the element name, in bytes (input).

NAME The element name (input).

ELEM The selected named element in the message (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTNC (get named element count)

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

CALL 'AMMSGTNC' USING HMSG, NAMELEN, NAME, COUNT, COMPCODE, REASON.

Declare the parameters as follows:

```
01 HMSG
                 PIC S9(9) BINARY.
                 PIC S9(9) BINARY.
01 NAMELEN
01 NAME
                 PIC X(n).
                 PIC S9(9) BINARY.
01 COUNT
                 PIC S9(9) BINARY.
01 COMPCODE
01 REASON
                 PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

NAMELEN The length of the element name, in bytes (input).

NAME The specified element name (input).

COUNT The number of elements in the message with the specified name

(output).

COMPCODE Completion code (output).

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). In addition to application defined

values, the following can be returned:

AMFB-NONE

AMFB-CODE-EXPIRATION AMFB-CODE-COA AMFB-CODE-COD

Error code AMRC_MSG_TYPE_NOT_REPORT may be issued.

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTTO (get topic)

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

```
CALL 'AMMSGTTO' USING HMSG, TOPICINDEX, BUFFLEN, TOPICLEN, TOPIC, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG PIC S9(9) BINARY.
01 TOPICINDEX PIC S9(9) BINARY.
01 BUFFLEN PIC S9(9) BINARY.
01 TOPICLEN PIC S9(9) BINARY.
01 TOPIC PIC X(n).
01 COMPCODE PIC S9(9) BINARY.
01 REASON PIC S9(9) BINARY.
```

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).

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 329). 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:

HMSG	PIC	S9(9)	BINARY.
READLEN	PIC	S9(9)	BINARY.
DATALEN	PIC	S9(9)	BINARY.
DATA	PIC	X(n).	
COMPCODE	PIC	S9(9)	BINARY.
REASON	PIC	S9(9)	BINARY.
	READLEN DATALEN DATA	READLEN PIC DATALEN PIC DATA PIC COMPCODE PIC	READLEN PIC S9(9) DATALEN PIC S9(9) DATA PIC X(n). COMPCODE PIC S9(9)

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 313).

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).

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.
```

The message handle returned by AMSECRMS (input).

CCSID The coded character set identifier (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSSTCI (set correl ID)

Sets the correlation identifier of the message.

```
CALL 'AMMSSTCI' USING HMSG, CORRELIDLEN, CORRELID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG PIC S9(9) BINARY.
01 CORRELIDLEN PIC S9(9) BINARY.
01 CORRELID PIC X(n).
01 COMPCODE PIC S9(9) BINARY.
01 REASON PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

CORRELIDLEN The length of the correlation identifier, in bytes (input).

CORRELID The correlation identifier (input). If CORRELIDLEN is set to zero,

the message correlation identifier is reset and the CORRELID

parameter will be ignored.

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSSELC (set element CCSID)

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

```
CALL 'AMMSSELC' USING HMSG, ELEMENTCCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

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

HMSG The message handle returned by AMSECRMS (input).

ELEMENTCCSID The element coded character set identifier (input).

COMPCODE Completion code (output).

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 327 and "AMMSWRBY (write bytes)" on page 331.

```
CALL 'AMMSSTDO' USING HMSG, OFFSET, COMPCODE, REASON.
```

Declare the parameters as follows:

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

The message handle returned by AMSECRMS (input).

OFFSET The offset in bytes (input). Set an offset of zero to read or write

from the start of the data.

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSSTEN (set encoding)

Sets the encoding of the data in the message.

```
CALL 'AMMSSTEN' USING HMSG, ENCODING, COMPCODE, REASON.
```

Declare the parameters as follows:

01	HMSG	PIC	S9(9)	BINARY.
01	ENCODING	PIC	S9(9)	BINARY.
01	COMPCODE	PIC	S9(9)	BINARY.
01	REASON	PIC	S9(9)	BINARY.

HMSG The message handle returned by AMSECRMS (input).

ENCODING The encoding of the message (input). It can take one of the

following values:

AMENC-NATIVE AMENC-NORMAL

AMENC-NORMAL-FLOAT-390

AMENC-REVERSED

AMENC-REVERSED-FLOAT-390

AMENC-UNDEFINED

COMPCODE Completion code (output).

AMMSSTFO (set format)

Sets the format of the message.

CALL 'AMMSSTFO' USING HMSG, FORMATLEN, FORMAT, COMPCODE, REASON.

Declare the parameters as follows:

```
01 HMSG PIC S9(9) BINARY.
01 FORMATLEN PIC S9(9) BINARY.
01 FORMAT PIC X(n).
01 COMPCODE PIC S9(9) BINARY.
01 REASON PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

FORMATLEN The length of the format, in bytes (input).

FORMAT The format of the message (input). It can take one of the following

values, or an application defined string:

AMFMT-NONE AMFMT-STRING AMFMT-RF-HEADER

If set to AMFMT-NONE, the default format for the sender will be

used (if available).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSSTGS (set group status)

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

If you specify AMGRP-MIDDLE-MSG-IN-GROUP or AMGRP-LAST-MSG-IN-GROUP without specifying AMGRP-FIRST-MSG-IN-GROUP, the behavior is the same as for AMGRP-FIRST-MSG-IN-GROUP and AMGRP-ONLY-MSG-IN-GROUP respectively.

If you specify AMGRP-FIRST-MSG-IN-GROUP out of sequence, the behavior is the same as for AMGRP-MIDDLE-MSG-IN-GROUP.

CALL 'AMMSSTGS' USING HMSG, STATUS, COMPCODE, REASON.

Declare the parameters as follows:

01	HMSG	PIC	S9(9)	BINARY.
01	STATUS	PIC	S9(9)	BINARY.
01	COMPCODE	PIC	S9(9)	BINARY.
01	REASON	PIC	S9(9)	BINARY.

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 329.

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, policies, and policy handlers" on page 491). Once created, it must not be opened until used in its correct context as a response sender when receiving a request message with AMRCRC (receive) or AMHRCRQ (receive request). When opened, its queue and queue manager properties are modified to reflect the ReplyTo destination specified in the message being received. When first used in this context, the sender service becomes a response sender service.

AMSNCLEC (clear error codes)

Clears the error codes in the sender object.

CALL 'AMSNCLEC' USING HSENDER, COMPCODE, REASON.

Declare the parameters as follows:

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

HSENDER The sender handle returned by AMSECRSN (input).

BUFFLEN Reserved, must be zero (input).

Reserved (output). **STRINGLEN 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:

```
PIC S9(9) BINARY.
01 HSENDER
01 BUFFLEN
                PIC S9(9) BINARY.
01 NAMELEN
                PIC S9(9) BINARY.
                PIC X(n).
01 NAME
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:

```
PIC S9(9) BINARY.
   HSENDER
   HPOLICY
                 PIC S9(9) BINARY.
                 PIC S9(9) BINARY.
01 OPTIONS
                 PIC S9(9) BINARY.
01 DIRNAMELEN
                 PIC X(n).
01 DIRNAME
                 PIC S9(9) BINARY.
01 FILENAMELEN
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 346 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-DEFAULT (AMBRW-NEXT)

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

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.
   HRCVMSG
01
   HRESPONSE
                 PIC S9(9) BINARY.
                 PIC S9(9) BINARY.
   COMPCODE
0.1
                 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

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 317)

before invoking the **AMRCBRSE** function.

BUFFLEN The length in bytes of a buffer in which the data is returned

(input).

DATALEN The length of the message data, in bytes (output). This can be set

to -1 (input).

DATA The received message data (output).

HRCVMSG The handle of the message object for the received message

(output).

HSENDER The handle of the response sender service that the response

> message must be sent to, if this is a request message (output). This sender service must be created without a repository definition (that is, it must not exist before the AMI session is started), and must be

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

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).

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

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 328) 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:

PIC S9(9) BINARY. 01 HRECEIVER PIC S9(9) BINARY. 01 NAMELEN 01 QUEUENAME PIC X(n). 01 COMPCODE PIC S9(9) BINARY. PIC S9(9) BINARY. 01 REASON

HRECEIVER The receiver handle returned by AMSECRRC (input).

NAMELEN The length of the queue name, in bytes (input). **QUEUENAME** The queue name of the receiver service (input).

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

Distribution list interface functions

A distribution list object encapsulates a list of sender objects.

AMDLCLEC (clear error codes)

```
Clears the error codes in the distribution list object.
```

```
CALL 'AMDLCLEC' USING HDISTLIST, COMPCODE, REASON.
```

Declare the parameters as follows:

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

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

COMPCODE Completion code (output).

REASON Reason code (output).

AMDLCL (close)

Closes the distribution list.

```
CALL 'AMDLCL' USING HDISTLIST, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

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

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

HPOLICY The handle of a policy (input). If specified as AMH-NULL-

HANDLE, the system default policy (constant:

AMSD-POL-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMDLGTLE (get last error)

Gets the information (completion and reason codes) from the last error in the distribution list object.

```
CALL 'AMDLGTLE' USING HDISTLIST, BUFFLEN, STRINGLEN, ERRORTEXT, REASON2, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HDISTLIST PIC S9(9) BINARY.
01 BUFFLEN PIC S9(9) BINARY.
01 STRINGLEN PIC S9(9) BINARY.
01 ERRORTEXT PIC X(n).
01 REASON2 PIC S9(9) BINARY.
01 COMPCODE PIC S9(9) BINARY.
01 REASON PIC S9(9) BINARY.
```

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

BUFFLEN Reserved, must be zero (input).

STRINGLEN Reserved (output).

ERRORTEXT Reserved (output).

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

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

HANDLEINDEX The index of the required sender service in the distribution list

(input). Specify an index of zero to return the first sender service in

the list.

Use AMDLGTSC to get the number of sender services in the

distribution list.

HSENDER The handle of the sender service (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMDLOP (open)

Opens the distribution list object for each of the destinations in the distribution list. The completion and reason codes returned by this function call indicate if the open was unsuccessful, partially successful, or completely successful.

CALL 'AMDLOP' USING HDISTLIST, HPOLICY, COMPCODE, REASON.

Declare the parameters as follows:

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

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

HPOLICY The handle of a policy (input). If specified as AMH-NULL-

HANDLE, the system default policy (constant:

AMSD-POL-HANDLE) is used.

COMPCODE Completion code (output).

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 346 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).

ENCODING The encoding (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMPBGTLE (get last error)

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

CALL 'AMPBGTLE' USING HPUBLISHER, BUFFLEN, STRINGLEN, ERRORTEXT, REASON2, COMPCODE, REASON.

Declare the parameters as follows:

```
01 HPUBLISHER PIC S9(9) BINARY.
01 BUFFLEN PIC S9(9) BINARY.
01 STRINGLEN PIC S9(9) BINARY.
01 ERRORTEXT PIC X(n).
01 REASON2 PIC S9(9) BINARY.
01 COMPCODE PIC S9(9) BINARY.
01 REASON PIC S9(9) BINARY.
```

HPUBLISHER The publisher handle returned by AMSECRPB (input).

BUFFLEN Reserved, must be zero (input).

STRINGLEN Reserved (output).

ERRORTEXT Reserved (output).

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

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

REASON2 gives 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).

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	PIC	S9(9)	BINARY.
01	HPOLICY	PIC	S9(9)	BINARY.
01	HRECEIVER	PIC	S9(9)	BINARY.
01	HPUBMSG	PIC	S9(9)	BINARY.
01	COMPCODE	PIC	S9(9)	BINARY.
01	REASON	PIC	S9(9)	BINARY.

HPUBLISHER The publisher handle returned by AMSECRPB (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-

HANDLE, the system default policy (constant:

AMSD-POL-HANDLE) is used.

HRECEIVER The handle of the receiver service to which the response to this

publish request should be sent (input). Specify as

AMH-NULL-HANDLE if no response is required. This parameter is mandatory if the policy specifies implicit registration of the

publisher.

HPUBMSG The handle of a message object for the publication message (input).

If specified as AMH-NULL-HANDLE, the default message object

(constant: AMSD-SND-MSG-HANDLE) is used.

COMPCODE Completion code (output).

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).

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 313).

The message data is passed in the message object. There is no option to pass it as a separate parameter as with AMRCRC (this would not give any performance improvement because the MQRFH header has to be removed from the message data after receiving it).

```
CALL 'AMSBRC' USING HSUBSCRIBER, HPOLICY, HSELMSG, HRCVMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

01	HSUBSCRIBER	PIC	S9(9)	BINARY.
01	HPOLICY	PIC	S9(9)	BINARY.
01	HSELMSG	PIC	S9(9)	BINARY.
01	HRCVMSG	PIC	S9(9)	BINARY.
01	COMPCODE	PIC	S9(9)	BINARY.
01	REASON	PIC	S9(9)	BINARY.

HSUBSCRIBER The subscriber handle returned by AMSECRSB (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-

HANDLE, the system default policy (constant:

AMSD-POL-HANDLE) is used.

HSELMSG The handle of a selection message object (input). This is used to

identify the message to be received (for example, using the correlation ID). Specify as AMH-NULL-HANDLE to get the next

available message with no selection.

HRCVMSG The handle of the message object for the received message (input).

If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-RCV-MSG-HANDLE) is used. The message object

is reset implicitly before the receive takes place.

COMPCODE Completion code (output).

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 313) before sending the message.

Publications matching the subscription are sent to the receiver service associated with the subscriber. By default, this has the same name as the subscriber service, with the addition of the suffix '.RECEIVER'.

```
CALL 'AMSBSB' USING HSUBSCRIBER, HPOLICY, HRECEIVER, HSUBMSG,
                    COMPCODE, REASON.
```

Declare the parameters as follows:

01	HSUBSCRIBER	PIC	S9(9)	BINARY.
01	HPOLICY	PIC	S9(9)	BINARY.
01	HRECEIVER	PIC	S9(9)	BINARY.
01	HSUBMSG	PIC	S9(9)	BINARY.
01	COMPCODE	PIC	S9(9)	BINARY.
01	REASON	PIC	S9(9)	BINARY.

HSUBSCRIBER The subscriber handle returned by AMSECRSB (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-

HANDLE, the system default policy (constant:

AMSD-POL-HANDLE) is used.

HRECEIVER The handle of the receiver service to which the response to this

> subscribe request should be sent (input). Specify as AMH-NULL-HANDLE if no response is required.

This is not the service to which publications will be sent by the broker; they are sent to the receiver service associated with the

subscriber (see above).

HSUBMSG The handle of a message object for the subscribe message (input).

If specified as AMH-NULL-HANDLE, the default message object

(constant: AMSD-SND-MSG-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMSBUN (unsubscribe)

Sends an unsubscribe message to a publish/subscribe broker using the subscriber service, to deregister a subscription. The topic and other elements can be specified using the message interface functions (see page 313) 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	HZORZCKIBEK	PIC	59(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 14. Using the Application Messaging	Send file
Interface in Java	71 Get values
Structure of the AMI	B71 Error handling
Base classes	
Interface and helper classes	Open and close
Exception classes	Receive and browse
Using the repository	Receive file
System default objects	
Writing applications in Java	
Creating and opening objects	
Sending messages	
Sample program	
Receiving messages	
Sample program	
Request/response messaging 3	
Sample programs	
File transfer	
Publish/subscribe messaging 3	
Sample programs	
Using AmElement objects	
Error handling	
Transaction support	
Sending group messages	
Other considerations	Broker messages
Multithreading	
Using MQSeries with the AMI	
Field limits	
Building Java applications	
AMI package for Java	
Running Java programs	882 Error handling
Running Java programs	Helper classes
Chapter 15. Java interface overview	
Base classes	
Helper classes	
Exception classes	
AmSessionFactory	
Constructor	<u>-</u>
Session factory management	
Create session	
AmSession	
Session management	Chapter 16. Java interface reference
Create objects	
Transactional processing	
Error handling	*
AmMessage	· · · · · · · · · · · · · · · · · · ·
Get values	
Set values	
Reset values	0 7
Read and write data	
Publish/subscribe filters	
Publish/subscribe topics	
Publish/subscribe name/value elements 3	
Error handling	
AmSender	
Open and close	
Send	setTraceLocation

AmSession	clearErrorCodes 413
begin	close
clearErrorCodes 402	enableWarnings 413
close	getCCSID
commit	getEncoding
createDistributionList	getLastErrorStatus
createMessage	getName
createPolicy	open
createPublisher 403	send
createReceiver 403	sendFile
createSender	AmReceiver
createSubscriber 403	browse
enableWarnings 404	clearErrorCodes 417
getLastErrorStatus	close
getName	
	ě
getTraceLevel	getDefinitionType
getTraceLocation	getLastErrorStatus
open	getName
rollback	getQueueName 418
AmMessage	open
addElement	receive
addFilter	receiveFile 419
addTopic	setQueueName 419
clearErrorCodes	AmDistributionList
deleteElement	clearErrorCodes
deleteFilter	close
deleteNamedElement	enableWarnings
deleteTopic	getLastErrorStatus
enableWarnings 407	getName
getCCSID	getSender
getCorrelationId 407	getSenderCount 420
getDataLength 407	open
getDataOffset	send
getElement	sendFile
getElementCount 407	AmPublisher
getEncoding	clearErrorCodes 422
getFilter	close
getFilterCount	enableWarnings
getFormat	getCCSID
0	0
getGroupStatus	getEncoding
getLastErrorStatus	getLastErrorStatus
getMessageId 409	getName
getName	open
getNamedElement 409	publish
getNamedElementCount 409	AmSubscriber
getReportCode	clearErrorCodes 424
getTopic	close
getTopicCount 409	enableWarnings 424
getType	getCCSID
readBytes	getDefinitionType
reset	0 ,1
	getEncoding
	getLastErrorStatus
setCorrelationId	getName
setDataOffset	getQueueName
setEncoding 411	open
setFormat 411	receive
setGroupStatus 411	setQueueName 425
setReportCode 411	subscribe
setType	unsubscribe
writeBytes 412	AmPolicy
AmSender	clearErrorCodes 427
	· · · · · · · · · · · · · · · · · · ·

	enableWarnings												427
	enableWarnings getLastErrorStatus												427
	getName												427
	getWaitTime setWaitTime nConstants												427
	setWaitTime												427
Ar	nConstants												428
Ar	nElement												429
	AmElement												429
	AmElement getName												429
	getValue												429
	getVersion												429
	setVersion												429
	setVersion toString												429
Ar	nObject												430
	clearErrorCodes												430
	clearErrorCodes getLastErrorStatus												430
	getName												430
Ar	nStatus												431
	AmStatus												431
	AmStatus getCompletionCod	le											431
	getReasonCode.												431
	getReasonCode2												431
	toString												431
Ar	toString nException												432
	getClassName .												432
	getCompletionCod	le											432
	getCompletionCod getMethodName												432
	getReasonCode.												432
	getSource												
	toString												432
Ar	toString nErrorException.												433
	getClassName .												433
	getCompletionCod	le											433
	getMethodName												433
	getMethodName getReasonCode.												433
	getSource												433
	toString												433
Ar	toString nWarningException	ı											434
	getClassName . getCompletionCod												434
	getCompletionCod	le											434
	getMethodName												434
	getMethodName getReasonCode.												434
	getSource												434
	Le Chaire	•	•	•	-	-	-	•	•	•	•	•	101

Chapter 14. Using the Application Messaging Interface in Java

The Application Messaging Interface for Java (amJava) provides a Java style of programming, while being consistent with the object-style interface of the Application Messaging Interface for C. It uses a Java Native Interface (JNI) library, so it cannot be used to write Applets to run in a browser environment.

This chapter describes the following:

- "Structure of the AMI"
- "Writing applications in Java" on page 373
- "Building Java applications" on page 382

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 492).

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

System default objects

The set of system default objects created in C is not accessible directly in Java, but the SYSTEM.DEFAULT.POLICY (constant: AMSD_POL) is used to provide default behavior when a policy is not specified. Objects with identical properties to the system default objects can be created for use in Java using the built-in definitions (see "System provided definitions" on page 492).

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 388.

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 410) 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 481).

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 Correl Id 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 481).

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 373, 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 375, 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 375).

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 481).

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 478).

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 481).

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.

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

com.ibm.mq.amt Java package com.ibm.mq.amt.jar Java jar file

This jar file is installed under:

/QIBM/ProdData/mgm/amt/Java/lib (AS/400)/java/lib (UNIX) \java\lib (Windows)

See "Directory structure" on page 445 (AIX), page 449 (AS/400), page 455 (HP-UX), page 463 (Solaris), or page 468 (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 445 (AIX), on page 448 (AS/400), page 454 (HP-UX), page 462 (Solaris), or page 467 (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, Windows NT, Windows Me, and Windows 2000 operating systems.

The AMI interface for Java makes use of JNI (Java Native Interface) and so requires a platform native library to run successfully. This library must be accessible to your runtime environment. See "Language compilers" on page 442 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 15. Java interface overview

This chapter contains an overview of the structure of the Application Messaging Interface for Java. Use it to find out what functions are available in this interface.

The Java interface provides sets of methods for each of the classes listed below. The methods available for each class are listed in the following pages. Follow the page references to see the reference information for each method.

Base classes

AmSessionFactory	page 386
AmSession	page 387
AmMessage	page 388
AmSender	page 390
AmReceiver	page 391
AmDistributionList	page 392
AmPublisher	page 393
AmSubscriber	page 394
AmPolicy	page 395

Helper classes

AmConstants	page 396
AmElement	page 396
AmObject	page 396
AmStatus	page 396

Exception classes

AmException	page 397
AmErrorException	page 397
AmWarningException	page 397

AmSessionFactory

The AmSessionFactory class is used to create AmSession objects.

Constructor

Constructor for AmSessionFactory.

AmSessionFactory page 400

Session factory management

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

getFactoryName	page 400
getLocalHost	page 400
getRepository	page 400
getTraceLevel	page 400
getTraceLocation	page 400
setLocalHost	page 400
setRepository	page 401
setTraceLevel	page 401
setTraceLocation	page 401

Create session

Method to create an AmSession object.

createSession page 400

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

page 404
page 402
page 404
page 404
page 404

Create objects

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

createMessage	page 403
createSender	page 403
createReceiver	page 403
create Distribution List	page 402
createPublisher	page 403
createSubscriber	page 403
createPolicy	page 403

Transactional processing

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

begin	page 402
commit	page 402
rollback	page 404

Error handling

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

clearErrorCodes	page 402
enableWarnings	page 404
getLastErrorStatus	page 404

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 407
getCorrelationId	page 407
getEncoding	page 408
getFormat	page 408
getGroupStatus	page 408
getMessageId	page 409
getName	page 409
getReportCode	page 409
getType	page 410

Set values

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

setCCSID	page 410
setCorrelationId	page 410
setEncoding	page 411
setFormat	page 411
set Group Status	page 411
setReportCode	page 411
setType	page 412

Reset values

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

reset page 410

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 407
getDataOffset	page 407
set Data Off set	page 410
readBytes	page 410
writeBytes	page 412

Publish/subscribe filters

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

addFilterpage 406deleteFilterpage 406getFilterpage 408getFilterCountpage 408

Publish/subscribe topics

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

addTopicpage 406deleteTopicpage 407getTopicpage 409getTopicCountpage 409

Publish/subscribe name/value elements

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

addElementpage 405deleteElementpage 406getElementpage 407getElementCountpage 407deleteNamedElementpage 406getNamedElementpage 409getNamedElementCountpage 409

Error handling

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

clearErrorCodespage 406enableWarningspage 407getLastErrorStatuspage 408

AmSender

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

Open and close

Methods to open and close the sender service.

page 414 open close page 413

Send

Method to send a message.

send page 414

Send file

Method to send data from a file

sendFile page 415

Get values

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

getCCSID page 413 getEncoding page 413 getName page 414

Error handling

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

clearErrorCodes page 413 enableWarnings page 413 getLastErrorStatuspage 414

AmReceiver

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

Open and close

Methods to open and close the receiver service.

open page 418 close page 417

Receive and browse

Methods to receive or browse a message.

receive page 418

browse page 416

Receive file

Method to receive file message data into a file.

receiveFile page 419

Get values

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

getDefinitionType page 417
getName page 418
getQueueName page 418

Set value

Method to set the queue name of the receiver service.

setQueueName page 419

Error handling

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

clearErrorCodespage 417enableWarningspage 417getLastErrorStatuspage 418

AmDistributionList

An AmDistributionList object encapsulates a list of AmSender objects.

Open and close

Methods to open and close the distribution list service.

open page 420 close page 420

Send

Method to send a message to the distribution list.

send page 421

Send file

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

sendFile page 421

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 420 getSenderCount page 420 getSender page 420

Error handling

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

clearErrorCodes page 420 enableWarnings page 420 getLastErrorStatus page 420

AmPublisher

An **AmPublisher** object encapsulates a sender service and provides support for publishing messages to a publish/subscribe broker.

Open and close

Methods to open and close the publisher service.

open page 423 close page 422

Publish

Method to publish a message.

publish page 423

Get values

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

getCCSID page 422
getEncoding page 422
getName page 422

Error handling

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

clearErrorCodespage 422enableWarningspage 422getLastErrorStatuspage 422

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 425 open close page 424

Broker messages

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

subscribe page 426 unsubscribe page 426 receive page 425

Get values

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

getCCSID page 424 getDefinitionType page 424 getEncoding page 424 getName page 425 getQueueName page 425

Set value

Method to set the queue name of the subscriber service.

setQueueName page 425

Error handling

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

clearErrorCodes page 424 enableWarnings page 424 getLastErrorStatus page 424

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 427getWaitTimepage 427setWaitTimepage 427

Error handling

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

clearErrorCodespage 427enableWarningspage 427getLastErrorStatuspage 427

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 428

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 429
getName	page 429
getValue	page 429
getVersion	page 429
setVersion	page 429
toString	page 429

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 430
getLastErrorStatus	page 430
getName	page 430

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 431
getCompletionCode	page 431
getReasonCode	page 431
getReasonCode2	page 431
toString	page 431

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 432
get Completion Code	page 432
get Method Name	page 432
getReasonCode	page 432
getSource	page 432
toString	page 432

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 433
get Completion Code	page 433
getMethodName	page 433
getReasonCode	page 433
getSource	page 433
toString	page 433

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 434
get Completion Code	page 434
getMethodName	page 434
getReasonCode	page 434
getSource	page 434
toString	page 434

Java interface overview

Chapter 16. Java interface reference

In the following sections the Java interface methods are listed by the class they refer to. Within each section the methods are listed in alphabetical order.

Note that where constants are shown (for example, AMRC_NONE), they can be accessed using the AmConstants class (for example, AmConstants.AMRC_NONE). See page 428.

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 400
AmSession	page 402
AmMessage	page 405
AmSender	page 413
AmReceiver	page 416
AmDistributionList	page 420
AmPublisher	page 422
AmSubscriber	page 424
AmPolicy	page 427

Helper classes

AmConstants	page 428
AmElement	page 429
AmObject	page 430
AmStatus	page 431

Exception classes

AmException	page 432
AmErrorException	page 433
AmWarningException	page 434

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 471 for the location of these files.

createSession

Creates an AmSession object.

AmSession createSession(String name);

name

The name of the AmSession.

getFactoryName

Returns the name of the AmSessionFactory.

String getFactoryName();

getLocalHost

Returns the name of the local host file.

String getLocalHost();

getRepository

Returns the name of the repository file.

String getRepository();

getTraceLevel

Returns the trace level for the AmSessionFactory.

int getTraceLevel();

getTraceLocation

Returns the location of the trace for the AmSessionFactory.

String getTraceLocation();

setLocalHost

Sets the name of the AMI local host file to be used by any AmSession created from this AmSessionFactory. (Otherwise, the default host file amthost.xml is used.)

void setLocalHost(String fileName);

fileName

The name of the file used by the AMI as the local host file. This file must be present on the local file system or an error will be produced upon the creation of an AmSession.

setRepository

Sets the name of the AMI repository to be used by any AmSession created from this AmSessionFactory. (Otherwise, the default repository file amt.xml is used.) void setRepository(String fileName);

fileName

Either of the following:

- The name of the file used by the AMI as the repository.

 This file must be present on the local file system or an error will be produced upon the creation of an AmSession.
- A reference to the repository information in LDAP URL format, when repository information is obtained from an LDAP directory.
 For details about specifying an LDAP URL, see "Directory search" on page 512.

setTraceLevel

Sets the trace level for the AmSessionFactory.

void setTraceLevel(int level);

level

The trace level to be set in the AmSessionFactory. Trace levels are 0 through 9, where 0 represents minimal tracing and 9 represents a fully detailed trace.

setTraceLocation

Sets the location of the trace for the AmSessionFactory.

void setTraceLocation(String location);

location

The location on the local system where trace files will be written. This location must be a directory, and it must exist before the trace is run.

AmSession

An AmSession object provides the scope for a unit of work and creates and manages all other objects, including at least one connection object. Each (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 410) and rebuild it each time.

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

addElement	page 405
deleteElement	page 406
getElement	page 407
getElementCount	page 407
delete Name d Element	page 406
getNamedElement	page 409
get Named Element Count	page 409
addTopic	page 406
deleteTopic	page 407
getTopic	page 409
getTopicCount	page 409

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);
```

The index of the element to be deleted, starting from zero. On completion, elements with higher index values than that specified will have those values reduced by one.

getElementCount gets the number of elements in the message.

deleteFilter

Deletes a publish/subscribe filter in an AmMessage object at the specified index. Indexing is within all filters in the message.

```
void deleteFilter(int filterIndex);
```

filterIndex

The index of the filter to be deleted, starting from zero. getFilterCount gets the number of filters in a message.

deleteNamedElement

Deletes the element with the specified name in the AmMessage object, at the specified index. Indexing is within all elements that share the same name.

```
void deleteNamedElement(
  String name,
  int
          index);
```

name

The name of the element to be deleted.

index

The index of the element to be deleted, starting from zero. On completion, elements with higher index values than that specified will have those values reduced by one.

getNamedElementCount gets the number of elements in the message with the specified name.

deleteTopic

Deletes a publish/subscribe topic in an AmMessage object at the specified index. Indexing is within all topics in the message.

```
void deleteTopic(int index);
```

index

The index of the topic to be deleted, starting from zero. getTopicCount gets the number of topics in the message.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using getLastErrorStatus, even if AmWarningExceptions are disabled.

void enableWarnings(boolean warningsOn);

If set to true, AmWarningExceptions will be raised for this object. warnings0n

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);
```

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.

```
int getNamedElementCount(String name);
```

name

The name of the elements to be counted.

getReportCode

Returns the feedback code from an AmMessage of type MQMT_REPORT. int getReportCode();

In addition to application defined values, the following values can be returned:

```
AMFB_NONE
AMFB_CODE_EXPIRATION
AMFB_CODE_COA
AMFB_CODE_COD
```

getTopic

Returns the publish/subscribe topic in the AmMessage object, at the specified index. Indexing is within all topics.

```
String getTopic(int index);
```

index

The index of the topic to be returned, starting from zero. **getTopicCount** gets the number of topics in the message.

getTopicCount

Returns the total number of publish/subscribe topics in the AmMessage object. int getTopicCount();

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 405).

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

options

A reserved field that must be set to zero.

setCCSID

Sets the coded character set identifier used by the AmMessage object.

```
void setCCSID(int codedCharSetId);
```

codedCharSetId

The CCSID to be set in the AmMessage.

setCorrelationId

Sets the correlation identifier in the AmMessage object.

```
void setCorrelationId(byte[] correlId);
```

correlId

The correlation identifier to be set in the AmMessage. The correlation identifier can be reset by specifying this as a zero length byte array. For example:

```
byte[] myByteArray = new byte[0];
myMessage.setCorrelationId(myByteArray);
```

setDataOffset

Sets the data offset for reading or writing byte data.

```
void setDataOffset(int dataOffset);
```

dataOffset

The data offset to be set in the AmMessage. Set an offset of zero to read or write from the start of the data.

setEncoding

Sets the encoding of the data in the AmMessage object.

void setEncoding(int encoding);

encoding

The encoding to be used in the AmMessage. It can take one of the following values:

AMENC_NORMAL

AMENC_NORMAL_FLOAT_390

AMENC REVERSED

AMENC REVERSED FLOAT 390

AMENC_UNDEFINED

setFormat

Sets the format for the AmMessage object.

void setFormat(String format);

format

The format to be used in the AmMessage. It can take one of the following values:

AMFMT_NONE AMFMT_STRING AMFMT_RF_HEADER

If set to AMFMT_NONE, the default format for the sender will be used (if available).

setGroupStatus

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

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

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

void setGroupStatus(int groupStatus);

groupStatus

The group status to be set in the AmMessage. It can take one of the following values:

AMGRP_MSG_NOT_IN_GROUP AMGRP_FIRST_MSG_IN_GROUP AMGRP_MIDDLE_MSG_IN_GROUP AMGRP_LAST_MSG_IN_GROUP AMGRP_ONLY_MSG_IN_GROUP

setReportCode

Sets the feedback code used by the AmMessage object. This is meaningful only for a message of type AMMT_REPORT.

void setReportCode(int reportCode);

reportCode

The feedback (or report code) value set in the AmMessage.

In addition to application defined values, the following values can be set:

AMFB NONE AMFB CODE EXPIRATION AMFB_CODE_COA AMFB CODE COD

setType

Sets the message type used by the AmMessage object. If a response message is requested with a publish, subscribe or unsubscribe request, the specified value is ignored and message type AMMT_REQUEST is used. If the value specified is AMMT_DATAGRAM, this is overridden when requesting or sending a response message (by AMMT_REQUEST and AMMT_RESPONSE, respectively).

void setType(int type);

type

The message type to be set in the AmMessage. It can take one of the following values:

AMMT DATAGRAM AMMT REQUEST AMMT REPLY AMMT REPORT

writeBytes

Writes a byte array into the AmMessage object, starting at the current data offset. If the data offset is not at the end of the data, existing data is overwritten. Use setDataOffset to specify the data offset. writeBytes will advance the data offset by the number of bytes written, leaving it immediately after the last byte written.

void writeBytes(byte[] data);

data

The data to be written to the AmMessage.

AmSender

An **AmSender** object encapsulates 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.

sendFile

Sends data from a file. To send data from a file, the sendMessage and fileName parameters are required, but the policy is optional. The file data can be received as normal message data by a target application using AmReceiver.receive, or used to reconstruct the file with AmReceiver.receiveFile.

void sendFile(
 AmMessage sendMessage,
 String filename,
 AmPolicy policy);

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

specify the Correlation ID for example.

fileName The name of the file to be sent (input). This can include a directory

prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel

with the message for use with the receive file method (see "receiveFile" on page 419 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 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 410) 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.

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 410) 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 410) 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.

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 AmPolicy policy);

sendMessage The message object containing the data to be sent.

responseService

The AmReceiver to be used for receiving any response to the sent

message. If omitted, no response can be received.

policy The policy to be used. If omitted, the system default policy

(constant: AMSD_POL) is used.

sendFile

Sends data from a file to each AmSender defined in the AmDistributionList object. The sendMessage and fileName parameters are required to send data from a file, but the policy is optional. The file data can be received as normal message data by a target application using AmReceiver.receive, or used to reconstruct the file with AmReceiver.receiveFile.

void sendFile(
 AmMessage sendMessage,
 String fileName,
 AmPolicy policy);

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

specify the Correlation ID, for example.

fileName The name of the file to be sent (input). This can include a directory

prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with the receive file method (see

"receiveFile" on page 419 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);

If set to true, AmWarningExceptions will be raised for this object. warnings0n

getCCSID

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

```
int getCCSID();
```

getEncoding

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

```
int getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmPublisher.

```
String getName();
```

open

Opens an AmPublisher service. **open** is overloaded: the policy parameter is optional.

```
void open(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (AMSD_POL) is used.

publish

Publishes a message using the AmPublisher. **publish** is overloaded: the pubMessage parameter is required, but the others are optional.

```
void publish(
AmMessage pubMessage,
AmReceiver responseService,
AmPolicy policy);
```

pubMessage The message object that contains the data to be published.

responseService

The AmReceiver to which the response to the publish request should be sent. Omit it if no response is required. This parameter is mandatory if the policy specifies implicit registration of the publisher.

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmSubscriber

An AmSubscriber object encapsulates both an AmSender and an AmReceiver. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

clearErrorCodes

```
Clears the error codes in the AmSubscriber.
 void clearErrorCodes();
```

close

```
Closes the AmSubscriber. close is overloaded: the policy parameter is optional.
 void close(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using getLastErrorStatus, even if AmWarningExceptions are disabled.

void enableWarnings(boolean warningsOn);

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();
```

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 405).

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.

Java AmSubscriber

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 405) 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 405) 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 and structures" on page 561.

For example, to use the constant AMRC_NONE (an AMI reason code), specify AmConstants.AMRC_NONE.

Note: Not all of the constants available in the C and C++ programming interfaces are available in Java, because they are not all appropriate in this language. For instance, AmConstants does not contain AMB_TRUE or AMB_FALSE, because the Java language has its own true and false constants and these are used by the AMI for Java.

AmElement

An AmElement object encapsulates a name/value pair which can be added to an AmMessage object.

AmElement

```
Constructor for an AmElement object.
 AmElement(String name, String value);
```

name The name of the element. 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();

Part 6. OS/390 Subsystems

Chapter 17. Writing applications for	0	3/3	90	
subsystems				. 437
Writing IMS applications using AMI.				. 437
Writing CICS applications using AMI				. 437
Writing batch applications using AMI				. 438
Writing RRS-batch applications using	ΑN	1 I		. 438
RRS availability				. 438

Chapter 17. 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 532 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^{TM} 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 18. Installation and sample programs		Setting the runtime environment 46	
Prerequisites		Directory structure (Windows) 46	68
Disk space		Local host and repository files (AS/400, UNIX®,	
Operating environments		and Windows) 47	
MQSeries environment		Default location 47	
Language compilers		Default names 47	
LDAP support		Overriding the default location and names 47	
Installation on AIX		Local host file 47	
Installation		Repository file 47	
Manual installation		Local host and repository files (OS/390) 47	
Using amtInstall		Batch, RRS-batch, IMS 47	
Removing the AMI		CICS	
Setting the runtime environment	. 445	Local host file 47	
Java programs		Repository file 47	
Directory structure (AIX)		Repository and local host caches 47	
Installation on AS/400	. 448	Generating caches 47	75
Setting the runtime environment for Java		Using a cache 47	75
programs		Cache generator and viewing the repository	
Directory structure (AS/400)		messages	
Library QMQMAMI		The AMI Administration Tool 47	
Members of QMQMAMI/H	. 449	Installation 47	77
Members of QMQMAMI/AMTMQSC	. 450	Operation	77
Members of QMQMAMI/QCSRC	. 450	Connecting to MQSeries	
Members of QMQMAMI/QSRVSRC	. 450	Using MQSeries Integrator Version 1 47	78
Members of QMQMAMI/QCPPSRC	. 450	Using MQSeries Publish/Subscribe 47	78
Library QMQMAMIPHL	. 450	Using MQSeries Integrator Version 2 47	78
IFS	. 450	Migrating to MQSeries Integrator V2 from V1	
Installation on HP-UX	. 453	and MQSeries Publish/Subscribe 48	80
Installation	. 453	Creating default MQSeries objects 48	80
Manual installation	. 453	The sample programs 48	81
Using amtInstall	. 453	Sample programs for AS/400, UNIX, and	
Removing the AMI	. 453	Windows	81
Setting the runtime environment	. 454	Running the AS/400, UNIX, and Windows	
Java programs	. 454	sample programs 48	82
Directory structure (HP-UX)	. 455	MQSeries objects 48	82
Installation on OS/390	. 458	Repository and host files 48	82
Installation	. 458	Running the publish/subscribe samples 48	82
Setting the runtime environment	. 458	Setting the runtime environment 48	
Batch and RRS-batch	. 458	Running the C and C++ samples 48	83
IMS		Running the Java samples 48	
CICS	. 458	Running the policy handler library sample 48	83
Unicode character conversion	. 458	Running the sample programs (AS/400) 48	84
Batch, RRS-batch, IMS	. 458	Sample programs for OS/390 48	
CICS	. 458	Running the sample programs (OS/390) 48	85
Directory structure (OS/390)	. 459	Building the sample programs 48	
Installation on Sun Solaris		MQSeries objects 48	
Installation	. 461	Repository and host files 48	
Manual installation		Running the publish/subscribe samples 48	
Using amtInstall		Setting the runtime environment 48	
Removing the AMI		File name input for the file transfer samples 48	
Setting the runtime environment		Running the batch samples 48	
Java programs		Running the CICS samples 48	
Directory structure (Solaris)		Running the IMS samples 48	
Installation on Windows		The AMI policy handler sample program	
Installation		(amtsphlr)	87
Removing the AMI		Required entry points	
O The state of the	-	1 / 1	

Invocation point functions		489
Chapter 19. Defining services, policies, and		
policy handlers		491
policy handlers		491
System provided definitions	•	492
System default objects		
Service definitions	•	494
Service definitions	•	494
Distribution list	•	
Distribution list	•	496
Subscriber	•	496
Publisher		496
Policy definitions		497
Initialization attributes		497
General attributes		498
Send attributes		498
Receive attributes		500
Subscribe attributes		
Publish attributes		
Handler attributes	•	503
Handler attributes	•	503
Policy handler definitions	•	504
Policy nandler attributes	•	504
Chapter 20. Lightweight Directory Access		
Protocol support		505
Getting Started With LDAP		505
SecureWay Directory		506
Active Directory		506
Active Directory		506
SecureWay Directory	-	507
OS/390 RDBM	•	508
Active Directory	•	508
Active Directory	•	
Uninstallation	•	509
Updating LDAP from a repository	•	509
Using the AMI Administration Iool	•	509
Using the command line		
Directory information tree		512
Directory search		512
Security		513
·		
Chapter 21. Problem determination		515
Using trace (AS/400, UNIX, and Windows)		
Trace filename and directory		
Commands on AS/400		
Commands on UNIX		
Community of UNIX	•	510
Commands on Windows		517
C++ and Java	•	518
Example trace		
Using trace (OS/390)		
Formatted Trace		529
Control of formatted trace		529
GTF Trace		530
Control of GTF Trace		530
When your AMI program fails		532
Reason Codes	•	532
First failure symptom report (AS/400, UNIX,		JJ2
		522
and Windows)	•	532
Other courses (is formally	•	33Z
Other sources of information		533
Common causes of problems		533

Chapter 18. Installation and sample programs

The Application Messaging Interface is available for the AIX, AS/400, HP-UX, OS/390, Sun Solaris, Windows 98, Windows NT, Windows Me, and Windows 2000 platforms.

This chapter contains:

- · "Prerequisites"
- "Installation on AIX" on page 444
- "Installation on AS/400" on page 448
- "Installation on HP-UX" on page 453
- "Installation on OS/390" on page 458
- "Installation on Sun Solaris" on page 461
- "Installation on Windows" on page 466
- "Local host and repository files (AS/400, UNIX®, and Windows)" on page 471
- "Local host and repository files (OS/390)" on page 473
- "The AMI Administration Tool" on page 477
- "Connecting to MQSeries" on page 478
- "The sample programs" on page 481

Prerequisites

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 17.3 MB AS/400 62.0 MB HP-UX 11.7 MB

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

Sun Solaris 12.7 MB

Windows 10.5 MB (without AMI Administration Tool)

54.5 MB (with AMI Administration Tool)

Operating environments

The AMI runs under the following operating systems:

AIX V4.3.3 (non-threaded and pthreads)

AS/400 V4R5 or later (non-threaded and pthreads)

HP-UX V11.0 (pthreads)

Prerequisites

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, V7, and V8 (pthreads)

Windows 98, Windows NT V4, Windows Me, and Windows 2000 Windows

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 5.2 or later
- MQSeries for AS/400 Version 5.2 or later
- MQSeries for HP-UX Version 5.2 or later
- MQSeries for OS/390 Version 5.2 or later
- MQSeries for Sun Solaris Version 5.2 or later
- MQSeries for Windows NT Version 5.2 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 5.2 or later
- MQSeries client for HP-UX Version 5.2 or later
- MQSeries client for Sun Solaris Version 5.2 or later
- MQSeries client for Windows NT and Windows 2000 Version 5.2 or later
- MQSeries client for Windows 95, 98 and Me Version 5.2 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 1.3 and later

AS/400 ILE C for AS/400 (5769CX2)

ILE C++ for AS/400 (5799GDW)

VisualAge C++ for OS/400 (5716CX4)

AS/400 Developer Kit for Java (5769JV1 option 5) i.e., JDK 1.3

HP-UX HP aC++ B3910B A.03.10

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

JDK 1.3 and later

Prerequisites

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 C++ 5.0

Forte C++ 6.0 JDK 1.3 and later

Windows Microsoft Visual C++ 6

JDK 1.3 and later

LDAP support

IBM SecureWay® Directory Version 3.2 Client or later

Next step

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

- "Installation on AIX" on page 444
- "Installation on AS/400" on page 448
- "Installation on HP-UX" on page 453
- "Installation on OS/390" on page 458
- "Installation on Sun Solaris" on page 461
- "Installation on Windows" on page 466

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 5.2, 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 445.

Removing the AMI

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

Setting the runtime environment

Make sure that the location of the AMI runtime binary files is added to your PATH environment variable. For example:

```
export PATH=$PATH:/usr/mqm/lib:
```

Note: The previous step is not needed if you used the amtInstall utility.

In addition, for the samples:

```
export PATH=$PATH:/usr/mqm/amt/samp/C/bin:/usr/mqm/amt/samp/Cpp/bin:
```

Java programs

When running Java, there are some additional steps.

The AMI classes must be contained in the CLASSPATH, for example: export CLASSPATH=\$CLASSPATH:/usr/mqm/java/lib/com.ibm.mq.amt.jar:

In addition, for the samples:

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 471 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/ipla : The International Program License Agreement file
./amt/li : The License Information file
./amt/inc
  amtc.h : The C header file for the AMI
  amtcpp.hpp: The C++ header file for the AMI
  amtphlc.h: The C header file for the policy handler interface
  amtphmqc.h: The C header file for the policy handler interface (for MQSeries)
  amtxc.h : The C header file for the extended AMI functions
  oamasami.h: The C header file for the OAMAS AMI subset
./amt/handlers
  libamtsphlr.a : C policy handler sample library
  libamtsphlr r.a : C policy handler sample threaded library
./amt/ldap/
  amtad.vbs : AMI Active Directory Visual Basic script file
  amtad.ldf: AMI Active Directory schema file
  amtsw.ldif: AMI SecureWay schema file for AIX, AS/400, HP-UX, Sun Solaris,
```

Installation on AIX

```
and Windows
   amtsw390.ldif : AMI SecureWay schema file for OS/390 TDBM
  amtsw390.at : AMI SecureWay schema file for OS/390 RDBM
  amtsw390.oc : AMI SecureWay schema file for OS/390 RDBM
./bin/amtldup : The AMI LDAP directory update program
./java/lib
  com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java
  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
  amtcmgic: Dynamic binding stub for MQSeries Client library
  amtcmqic r : Dynamic binding stub for MQSeries Client threaded library
  amtldap.a : The AMI LDAP library
  amtldap_r.a : The AMI LDAP 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
  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
```

Installation on AIX

```
amtshsvr : C high-level server sample program
  amtshpub : C high-level publisher sample program
  amtshsub : C high-level subscriber sample program
  amtshfsn : C high-level send file sample program
  amtshfrc : C high-level receive file sample program
./amt/samp/Cpp
  SendForget.cpp : C++ source for send and forget sample
  Receiver.cpp : C++ source for receiver sample
  Client.cpp : C++ source for client sample
  Server.cpp : C++ source for server sample
  Publisher.cpp : C++ source for publisher sample
  Subscriber.cpp : C++ source for subscriber sample
  RcvFile.cpp : C++ source for receive file sample
  SendFile.cpp : C++ source for send file sample
./amt/samp/Cpp/bin
  SendForget : C++ send and forget sample program
  Receiver: C++ receiver sample program
  Client : C++ client sample program
  Server : C++ server sample program
  Publisher: C++ publisher sample program
  Subscriber: C++ subscriber sample program
  RcvFile : C++ source for receive file sample
  SendFile : C++ source for send file sample
./amt/samp/C/handlers
  amtsphlh.h : C header file for policy handler sample
  amtsphlr.c : C source for policy handler sample
  amtsphlr.exp : C export file for policy handler sample
./amt/samp/java
  SendForget.java : Java source for send and forget sample
  Receiver.java : Java source for receiver sample
  Client.java : Java source for client sample
  Server.java : Java source for server sample
  Publisher.java : Java source for publisher sample
  Subscriber.java : Java source for subscriber sample
  RcvFile.java : Java source for receive file sample
  SendFile.java : Java source for send file sample
./amt/samp/java/bin
  com.ibm.mq.amt.samples.jar: The jar file containing the AMI
    samples class files for Java
```

Installation on AS/400

The AMI package for AS/400 comes as a compressed zip file ma0f 400.zip. Uncompress and restore it as follows:

- 1. Download ma0f 400.zip to a directory on your PC.
- 2. Uncompress the file using the InfoZip Unzip program.
 - The file ma0f 400.sav is created.
- 3. Create a save file called MA0F in a suitable library on the AS/400, for example the library QGPL:
 - CRTSAVF FILE (QGPL/MAOF)
- 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
- 5. Install the MQSeries for AS/400 AMI, product Id 5724A23, using RSTLICPGM: RSTLICPGM LICPGM(5724A23) DEV(*SAVF) SAVF(QGPL/MAOF)

If the Primary Language Feature ID of the system on which you are you are installing is not 2924, you must specify 2924 for the LNG option: RSTLICPGM LICPGM(5724A23) DEV(*SAVF) LNG(2924) SAVF(QGPL/MA0F)

6. Delete the save file created in Step 3:

command to create one. For example:

DLTF FILE (QGPL/MAOF)

To remove the AMI package from the AS/400, use DLTLICPGM: DLTLICPGM LICPGM(5724A23)

Setting the runtime environment for Java programs

To run Java:

- Ensure that the library QMQMAMI is in the library list. To add a library, you can use the ADDLIBLE command.
- Ensure that the AMI classes are contained in the CLASSPATH.

Use the WRKENVVAR command to determine whether a CLASSPATH exists. If a CLASSPATH environment variable does not exist, use the ADDENVVAR

ADDENVVAR ENVVAR(CLASSPATH) VALUE('/QIBM/ProdData/mqm/amt/java/lib/com.ibm.mq.amt.jar')

If a CLASSPATH environment variable already exists, use the CHGENVVAR command to add the following to it:

:/QIBM/ProdData/mqm/amt/java/lib/com.ibm.mq.amt.jar

To use the supplied samples, include the following in the CLASSPATH:

:/QIBM/ProdData/mgm/amt/samp/java/bin/com.ibm.mg.amt.samples.jar

Note: Each CLASSPATH entry must be separated by a colon.

Now go to "Local host and repository files (AS/400, UNIX®, and Windows)" on page 471 to continue the installation procedure.

Directory structure (AS/400)

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
AMTJAVA
            *SRVPGM
            *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
AMTSOPUB
            *PGM
                       : C object-level publisher sample program
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
                       : C object-level send and forget sample program
AMTSOSND
            *PGM
AMTSOSUB
            *PGM
                       : C object-level subscriber sample program
AMTSOSVR
            *PGM
                       : C object-level server sample program
            *PGM
                       : C++ client sample program
CLIFNT
            *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
                      : C++ server sample program
            *PGM
SFRVFR
                      : C++ subscriber sample program
: C sample files
: C++ samples
: Installation Exit Program
SUBSCRIBER *PGM
QCSRC
            *FILE
QCPPSRC
            *FILE
AMTIOX1C
            *PGM
AMTI0X0C
            *PGM
                      : Installation Exit Program
            *FILE
                      : Export file for policy handler sample
OSRVSRC
AMTLDAP
            *SRVPGM
                      : The AMI LDAP library
            *SRVPGM
AMTLDAP R
                       : The AMI LDAP threaded library
AMTLDUP
            *PGM
                        : The AMI LDAP directory update program
```

Members of QMQMAMI/H

```
AMTC : The C header file for AMI

AMTCPP : The C++ header file for AMI

AMTPHLC : Policy handler interface definition

AMTPHMQC : Transport-specific policy handler definitions for MQSeries

AMTSPHLH : The C header file for policy handler sample

AMTXC : AMI extensions for policy handler callback functions

OAMASAMI : The C header file for the OAMAS AMI subset
```

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

Members of QMQMAMI/AMTMQSC

AMTSDFTS : 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
amtshfrc
amtshfsn
amtshpub
amtshrv
amtshrv
amtshpub
amtshrv
amtshrv
amtshpub
amtshrv
amtshsnd
               : C source for high-level send and forget sample program
amtshsub
               : C source for high-level subscriber sample program
amtshsvr
               : C source for high-level server sample program
amtsoclt
              : C source for object-level client sample program
amtsofrc
              : C source for object-level receive file sample program
amtsofsn : C source for object-level send file sample program amtsopub : C source for object-level publisher sample program amtsorcv : C source for object-level receiver sample program
               : C source for object-level send file sample program
              : C source for object-level receive group sample program
amtsosgr
amtsosgs
               : C source for object-level send group sample program
amtsosnd
               : C source for object-level send and forget sample program
amtsosub : C source for object-level subscriber sample program
amtsosvr
               : C source for object-level server sample program
amtsphlr
               : C source for policy handler sample
```

Note: The members of the file QCSRC are copies of the C source files for sample programs in IFS.

Members of QMQMAMI/QSRVSRC

```
amtsphlr : Export file for policy handler sample
```

Note: The member of the file QSRVSRC is a copy of the export file for the policy handler sample in IFS.

Members of QMQMAMI/QCPPSRC

```
Client : C++ source for client sample
Publisher : C++ source for publisher sample
Receiver : C++ source for receiver sample
RcvFile : C++ source for receive file sample
SendForget : C++ source for send and forget sample
SendFile : C++ source for send file sample
Server : C++ source for server sample
Subscriber : C++ source for subscriber sample
```

Note: The members of the file QCPPSRC are copies of the C++ source files for sample programs in IFS.

Library QMQMAMIPHL

```
AMTSPHLR *SRVPGM : Sample policy handler library
```

AMTSPHLR R *SRVPGM : Sample policy handler threaded library

this documentation was produced

IFS

```
/QIBM/ProdData/mqm/amt
```

```
amt.dtd: Document Type Definition file on which the AMI repository is based amthost.xml: Sample AMI XML file used as the default host file amtsdfts.tst: MQSeries mqsc command file to create default MQSeries objects required by the AMI
ipla: The International Program License Agreement file
li: The License Information file
readme: Product and information updates that became available after
```

Installation on AS/400

```
/QIBM/ProdData/mgm/amt/inc
               : The C header file for AMI
    amtc.h
    amtcpp.hpp : The C++ header file for AMI
    amtphlc.h : The C header file for the policy handler interface
    amtphmgc.h : The C header file for the policy handler interface (for MOSeries)
              : The C header file for the extended AMI functions
    amtxc.h
    oamasami.h : The C header file for the OAMAS AMI subset
 /OIBM/ProdData/mgm/amt/samp
    amtsamp.tst: MQSeries mgsc command file to create MQSeries objects
                  required by AMI samples
                 : Sample AMI XML repository for use with the AMI samples \,
    amt.xml
 /QIBM/ProdData/mgm/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
    \verb|amtshpub.c| : C source for high-level subscriber sample program|\\
    amtshrcv.c : C source for high-level receiver sample program
amtshsnd.c : C source for high-level send and forget sample program
    amtshsub.c : C source for high-level subscriber sample program
    amtshsvr.c : C source for high-level server sample program
    amtsoclt.c : C source for object-level client sample program
    amtsofrc.c : C source for object-level receive file sample program
    amtsofsn.c : C source for object-level send file sample program
    amtsopub.c : C source for object-level publisher sample program
    amtsorcv.c : C source for object-level receiver sample program
    amtsosgr.c : C source for object-level receive group sample program
    amtsosgs.c : C source for object-level send group sample program
    amtsosnd.c : C source for object-level send and forget sample program
    amtsosub.c : C source for object-level subscriber sample program
    amtsosvr.c : C source for object-level server sample program
/QIBM/ProdData/mqm/amt/samp/C/handlers
    amtsphlr.c : \hat{\textbf{C}} source for policy handler sample
    amtsphlr.h
                       : C include for policy handler sample
    amtsphlr.exp
                       : Export file for the sample policy handler
 /QIBM/ProdData/mgm/amt/samp/Cpp
    Client.cpp : C++ source for client sample
    Publisher.cpp
                       : C++ source for publisher sample
    Receiver.cpp
                      : C++ source for receiver sample
    RcvFile.cpp
                      : C++ source for receive file sample
    SendForget.cpp
                      : C++ source for send and forget sample
    SendFile.cpp
                       : C++ source for send file sample
    Server.cpp
                       : C++ source for server sample
    Subscriber.cpp
                       : C++ source for subscriber sample
/QIBM/ProdData/mqm/amt/samp/java
    Client.java : Java source for client sample
    Publisher.java
                       : Java source for publisher sample
    Receiver.java : Java source for receiver sample
                      : Java source for receive file sample
    RcvFile.java
                       : Java source for send and forget sample
    SendForget.java
    SendFile.java
                       : Java source for send file sample
                       : Java source for server sample
    Server.java
    Subscriber.java : Java source for subscriber sample
 /QIBM/ProdData/mgm/amt/samp/java/bin
    com.ibm.mg.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
 /QIBM/ProdData/mgm/amt/ldap
```

Installation on AS/400

amtad.vbs :AMI Active Directory Visual Basic script file amtad.ldf :AMI Active Directory schema file amtsw.ldif :AMI SecureWay schema file for

AIX, AS/400, HP-UX, Sun Solaris and Windows

amtsw390.ldif :AMI SecureWay schema file for OS/390 TDBM amtsw390.at :AMI SecureWay schema file for OS/390 RDBM amtsw390.oc :AMI SecureWay schema file for OS/390 RDBM

/QIBM/UserData/mqm/amt

amt.dtd : Document Type Definition file on which the AMI repository is based

: Sample AMI XML file used as the default host file

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 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. 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 455.

Removing the AMI

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

Installation on HP-UX

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:

Java programs

When running Java, there are some additional steps.

The AMI classes must be contained in the CLASSPATH, for example: export CLASSPATH=\$CLASSPATH:/opt/mqm/java/lib/com.ibm.mq.amt.jar:

In addition, for the samples:

export CLASSPATH=\$CLASSPATH:/opt/mqm/amt/samp/java/bin /com.ibm.mq.amt.samples.jar:

Also, to load the AMI library for Java:

export SHLIB PATH=\$SHLIB PATH:/opt/mqm/lib:

Next step

Now go to "Local host and repository files (AS/400, UNIX®, and Windows)" on page 471 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/ipla : The International Program License Agreement file
./amt/li : The License Information file
./amt/inc
  amtc.h : The C header file for the AMI
  amtcpp.hpp: The C++ header file for the AMI
  amtphlc.h: The C header file for the policy handler interface
  amtphmqc.h: The C header file for the policy handler interface (for MQSeries)
  amtxc.h : The C header file for the extended AMI functions
  oamasami.h : The C header file for the OAMAS AMI subset
./amt/handlers
  libamtsphlr r.sl : C policy handler sample library
./amt/ldap/
  amtad.vbs : AMI Active Directory Visual Basic script file
  amtad.ldf : AMI Active Directory schema file
  amtsw.ldif: AMI SecureWay schema file for AIX, AS/400, HP-UX, Sun Solaris,
    and Windows
  amtsw390.ldif : AMI SecureWay schema file for OS/390 TDBM
  amtsw390.at: AMI SecureWay schema file for OS/390 RDBM
  amtsw390.oc : AMI SecureWay schema file for OS/390 RDBM
./bin/amtldup : The AMI LDAP directory update program
./java/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.
  amtcmqm r : Dynamic binding stub for MQSeries Server threaded library
  amtcmqic r : Dynamic binding stub for MQSeries Client threaded library
  libamtldap r.sl : The AMI LDAP 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
```

Installation on HP-UX

```
amtshsnd.c : C source for high-level send and forget sample
  amtshrcv.c : C source for high-level receiver sample
  amtshclt.c : C source for high-level client sample
  amtshsvr.c : C source for high-level server sample
  amtshpub.c : C source for high-level publisher sample
  amtshsub.c : C source for high-level subscriber sample
  amtshfsn.c : C source for high-level send file sample
  amtshfrc.c : C source for high-level receive file sample
./amt/samp/C/bin
  amtsosnd : C object-level send and forget sample program
  amtsorcv : C object-level receiver sample program
  amtsoclt : C object-level client sample program
  amtsosvr : C object-level server sample program
  amtsopub : C object-level publisher sample program
  amtsosub : C object-level subscriber sample program
  amtsofsn : C object-level send file sample program
  amtsofrc : C object-level receive file sample program
  amtsosgs : C object-level send group sample program
  amtsosgr : C object-level receive group sample program
  amtshsnd : C high-level send and forget sample program
  amtshrcv : C high-level receiver sample program
  amtshclt : C high-level client sample program
  amtshsvr : C high-level server sample program
  amtshpub : C high-level publisher sample program
  amtshsub : C high-level subscriber sample program
  amtshfsn : C high-level send file sample program
  amtshfrc : C high-level receive file sample program
./amt/samp/Cpp
  SendForget.cpp : C++ source for send and forget sample
  Receiver.cpp : C++ source for receiver sample
  Client.cpp : C++ source for client sample
  Server.cpp : C++ source for server sample
  Publisher.cpp : C++ source for publisher sample
  Subscriber.cpp : C++ source for subscriber sample
  RcvFile.cpp : C++ source for receive file sample
  SendFile.cpp : C++ source for send file sample
./amt/samp/Cpp/bin
  SendForget : C++ send and forget sample program
  Receiver: C++ receiver sample program
  Client: C++ client sample program
  Server : C++ server sample program
  Publisher: C++ publisher sample program
  Subscriber: C++ subscriber sample program
  RcvFile: C++ source for receive file sample
  SendFile : C++ source for send file sample
./amt/samp/C/handlers
  amtsphlh.h : C header file for policy handler sample
  amtsphlr.c : C source for policy handler sample
  amtsphlr.exp : C export file for policy handler sample
./amt/samp/iava
  SendForget.java : Java source for send and forget sample
  Receiver.java : Java source for receiver sample
  Client.java : Java source for client sample
  Server.java : Java source for server sample
  Publisher.java : Java source for publisher sample
  Subscriber.java : Java source for subscriber sample
  RcvFile.java : Java source for receive file sample
  SendFile.java : Java source for send file sample
```

Installation on HP-UX

./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 or later.

Installation

The files and directories created are listed in "Directory structure (OS/390)" on page 459.

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 473 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
hlg.SCSQANLK
  AMTMSG10 : Kanji messages
  AMTMSK10 : Kanji messages
hlg.SCSQANLC
  AMTMSG10 : Chinese messages
  AMTMSC10 : Chinese messages
hlq.SCSQC370
  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
          : The main COBOL copybook for the AMI
hlg.SCSQPROC
         : 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)
hlq.SCSQCOBS (COBOL samples for Batch, RRS, CICS, and IMS)
  AMTVHSND: COBOL source for high-level send and forget sample
  AMTVHRCV: COBOL source for high-level receiver sample
  AMTVHCLT: COBOL source for high-level client sample
  AMTVHSVR : COBOL source for high-level server sample
  AMTVHPUB : COBOL source for high-level publisher sample
  AMTVHSUB : COBOL source for high-level subscriber sample
```

Installation on OS/390

```
AMTVHFSN: COBOL source for high-level group send file transfer sample
 AMTVHFRC: COBOL source for high-level group receive file transfer sample
 AMTVOSND : COBOL source for object-level send and forget sample
 AMTVORCV : COBOL source for object-level receiver sample
 AMTVOCLT : COBOL source for object-level client sample
 AMTVOSVR : COBOL source for object-level server sample
 AMTVOPUB : COBOL source for object-level publisher sample
 AMTVOSUB : COBOL source for object-level subscriber sample
 AMTVOSGS : COBOL source for object-level group send sample
 AMTVOSGR : COBOL source for object-level group receive sample
 AMTVOFSN: COBOL source for object-level send file transfer sample
 AMTVOFRC: COBOL source for object-level receive file transfer sample
hlq.SCSQC37S (C samples for Batch, RRS, CICS, and IMS)
 AMTSHSND : C source for high-level send and forget sample
 AMTSHRCV: C source for high-level receiver sample
 AMTSHCLT: C source for high-level client sample
 AMTSHSVR : C source for high-level server sample
 AMTSHPUB : C source for high-level publisher sample
 AMTSHSUB : C source for high-level subscriber sample
 AMTSHFSN: C source for high-level group send file transfer sample
 AMTSHFRC: C source for high-level group receive file transfer sample
 AMTSOSND : C source for object-level send and forget sample
 AMTSORCV : C source for object-level receiver sample
 AMTSOCLT : C source for object-level client sample
 AMTSOSVR : C source for object-level server sample
 AMTSOPUB : C source for object-level publisher sample
 AMTSOSUB : C source for object-level subscriber sample
 AMTSOSGS : C source for object-level group send sample
 AMTSOSGR : C source for object-level group receive sample
 AMTSOFSN : C source for object-level send file transfer sample
 AMTSOFRC : C source for object-level receive file transfer sample
```

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
- 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 463.

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:

```
export CLASSPATH=$CLASSPATH:/opt/mqm/amt/samp/java/bin
                   /com.ibm.mq.amt.samples.jar:
```

Also, to load the AMI library for Java:

export LD LIBRARY PATH=\$LD LIBRARY PATH:/opt/mqm/lib:

Next step

Now go to "Local host and repository files (AS/400, UNIX®, and Windows)" on page 471 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/ipla : The International Program License Agreement file
./amt/li : The License Information file
./amt/inc
  amtc.h : The C header file for the AMI
  amtcpp.hpp: The C++ header file for the AMI
  amtphlc.h: The C header file for the policy handler interface
  amtphmqc.h: The C header file for the policy handler interface (for MQSeries)
  amtxc.h : The C header file for the extended AMI functions
  oamasami.h : The C header file for the OAMAS AMI subset
./amt/handlers
  libamtsphlr.so : C policy handler sample library
./amt/ldap/
  amtad.vbs : AMI Active Directory Visual Basic script file
  amtad.ldf : AMI Active Directory schema file
  amtsw.ldif: AMI SecureWay schema file for AIX, AS/400, HP-UX, Sun Solaris,
    and Windows
  amtsw390.ldif : AMI SecureWay schema file for OS/390 TDBM
  amtsw390.at: AMI SecureWay schema file for OS/390 RDBM
  amtsw390.oc : AMI SecureWay schema file for OS/390 RDBM
./bin/amtldup : The AMI LDAP directory update program
./java/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
  amtcmgm : Dynamic binding stub for MQSeries Server library
  amtcmgic: Dynamic binding stub for MQSeries Client library
  libamtldap r.so : The AMI LDAP 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
  \verb|amtsosgs.c|: C source for object-level send group sample
  amtsosgr.c : C source for object-level receive group sample
```

Installation on Sun Solaris

```
amtshsnd.c : C source for high-level send and forget sample
  amtshrcv.c : C source for high-level receiver sample
  amtshclt.c : C source for high-level client sample
  amtshsvr.c : C source for high-level server sample
  amtshpub.c : C source for high-level publisher sample
  amtshsub.c : C source for high-level subscriber sample
  amtshfsn.c : C source for high-level send file sample
  amtshfrc.c : C source for high-level receive file sample
./amt/samp/C/bin
  amtsosnd : C object-level send and forget sample program
  amtsorcv : C object-level receiver sample program
  amtsoclt : C object-level client sample program
  amtsosvr : C object-level server sample program
  amtsopub : C object-level publisher sample program
  amtsosub : C object-level subscriber sample program
  amtsofsn : C object-level send file sample program
  amtsofrc : C object-level receive file sample program
  amtsosgs : C object-level send group sample program
  amtsosgr : C object-level receive group sample program
  amtshsnd : C high-level send and forget sample program
  amtshrcv : C high-level receiver sample program
  amtshclt : C high-level client sample program
  amtshsvr : C high-level server sample program
  amtshpub : C high-level publisher sample program
  amtshsub : C high-level subscriber sample program
  amtshfsn : C high-level send file sample program
  amtshfrc : C high-level receive file sample program
 ./amt/samp/C/handlers
  amtsphlh.h : C header file for policy handler sample
  amtsphlr.c : C source for policy handler sample
  amtsphlr.exp : C export file for policy handler sample
./amt/samp/Cpp
  SendForget.cpp : C++ source for send and forget sample
  Receiver.cpp : C++ source for receiver sample
  Client.cpp : C++ source for client sample
  Server.cpp : C++ source for server sample
  Publisher.cpp : C++ source for publisher sample
  Subscriber.cpp : C++ source for subscriber sample
  RcvFile.cpp : C++ source for receive file sample
  SendFile.cpp : C++ source for send file sample
./amt/samp/Cpp/bin
  SendForget : C++ send and forget sample program
  Receiver: C++ receiver sample program
  Client : C++ client sample program
  Server : C++ server sample program
  Publisher: C++ publisher sample program
  Subscriber: C++ subscriber sample program
  RcvFile : C++ source for receive file sample
  SendFile : C++ source for send file sample
./amt/samp/iava
  SendForget.java : Java source for send and forget sample
  Receiver.java : Java source for receiver sample
  Client.java : Java source for client sample
  Server.java : Java source for server sample
  Publisher.java : Java source for publisher sample
  Subscriber.java : Java source for subscriber sample
  RcvFile.java : Java source for receive file sample
  SendFile.java : Java source for send file sample
```

Installation on Sun Solaris

./amt/samp/java/bin
com.ibm.mq.amt.samples.jar : The jar file containing the AMI
samples class files for Java

Installation on Windows

The AMI package for Windows 98, Windows NT, Windows Me and Windows 2000 comes as a zip file, ma0f nt.zip. Once unzipped it comprises:

A file containing any product and information updates that have

become available since this documentation was produced

InstallShield installation program for MQSeries AMI setup

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 468.

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" on page 466.)

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 471 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.dtd : AMI Document Type Definition file on which the AMI repository is based .\amt\ipla : The International Program License Agreement file .\amt\li : The License Information file .\amt\include amtc.h : The C header file for the AMI amtcpp.hpp : The C++ header file for the AMI amtphlc.h: The C header file for the policy handler interface amtphmqc.h: The C header file for the policy handler interface (for MQSeries) amtxc.h : The C header file for the extended AMI functions oamasami.h: The C header file for the OAMAS AMI subset .\amt\handlers amtsphlr.dll : C policy handler sample library .\amt\ldap\ amtad.vbs : AMI Active Directory Visual Basic script file amtad.ldf : AMI Active Directory schema file amtsw.ldif: AMI SecureWay schema file for AIX, AS/400, HP-UX, Sun Solaris, and Windows amtsw390.ldif: AMI SecureWay schema file for OS/390 TDBM amtsw390.at: AMI SecureWay schema file for OS/390 RDBM amtsw390.oc : AMI SecureWay schema file for OS/390 RDBM .\java\lib com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java amt.dll : The main AMI library amtXML310.dll : The AMI XML parsing library amtCpp.dll : The AMI C++ library amtJava.dll: The AMI JNI library amtICUUC140.dll : The AMI codepage translation library amtICUDATA.dll: The AMI codepage translation data library amtldap.dll : The AMI LDAP library

```
amt.lib : The AMI LIB file used for building C programs
  amtCpp.lib : The AMI LIB file used for building C++ programs
.\amt\samples
  amtsamp.tst: MQSeries mgsc 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

amtldup.exe : The AMI LDAP directory update program

.tools\lib

Installation on Windows

```
amtsosgr.c : C source for object-level receive group sample
  amtshsnd.c : C source for high-level send and forget sample
  amtshrcv.c : C source for high-level receiver sample
  amtshclt.c : C source for high-level client sample
  amtshsvr.c : C source for high-level server sample
  amtshpub.c : C source for high-level publisher sample
  amtshsub.c : C source for high-level subscriber sample
  amtshfsn.c : C source for high-level send file sample
  amtshfrc.c : C source for high-level receive file sample
.\amt\samples\C\bin
  amtsosnd.exe : C object-level send and forget sample program
  amtsorcv.exe : C object-level receiver sample program
  amtsoclt.exe : C object-level client sample program
  amtsosvr.exe : C object-level server sample program
  amtsopub.exe : C object-level publisher sample program
  amtsosub.exe : C object-level subscriber sample program
  amtsofsn.exe : C object-level send file sample program
  amtsofrc.exe : C object-level receive file sample program
  amtsosgs.exe : C object-level send group sample program
  amtsosgr.exe : C object-level receive group sample program
  amtshsnd.exe : C high-level send and forget sample program
  amtshrcv.exe : C high-level receiver sample program
  amtshclt.exe : C high-level client sample program
  amtshsvr.exe : C high-level server sample program
  amtshpub.exe : C high-level publisher sample program
  amtshsub.exe : C high-level subscriber sample program
  amtshfsn.exe : C high-level send file sample program
  amtshfrc.exe : C high-level receive file sample program
.\amt\samples\Cpp
  SendForget.cpp : C++ source for send and forget sample
  Receiver.cpp : C++ source for receiver sample
  Client.cpp : C++ source for client sample
  Server.cpp : C++ source for server sample
  Publisher.cpp : C++ source for publisher sample
  Subscriber.cpp : C++ source for subscriber sample
  RcvFile.cpp : C++ source for receive file sample
  SendFile.cpp : C++ source for send file sample
.\amt\samples\Cpp\bin
  SendForget.exe : C++ send and forget sample program
  Receiver.exe : C++ receiver sample program
  Client.exe : C++ client sample program
  Server.exe : C++ server sample program
  Publisher.exe : C++ publisher sample program
  Subscriber.exe : C++ subscriber sample program
  RcvFile.exe : C++ receive file sample program
  SendFile.exe : C++ send file sample program
.\amt\samples\handlers
  amtsphlh.h : C header file for policy handler sample
  amtsphlr.c : C source for policy handler sample
  amtsphlr.def : C definition file for policy handler sample
.\amt\samples\java
  SendForget.java : Java source for send and forget sample
  Receiver.java : Java source for receiver sample
  Client.java : Java source for client sample
  Server.java : Java source for server sample
  Publisher.java : Java source for publisher sample
  Subscriber.java : Java source for subscriber sample
  RcvFile.java : Java source for receive file sample
  SendFile.java : Java source for send file sample
```

Installation on Windows

.\amt\samples\java\bin
 com.ibm.mq.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/mqm/amt

On UNIX, the default directory for the files is:

/usr/mqm/amt (AIX)

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

On Windows, the default location is a directory called \amt under the 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 is:

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)
/amt/samp (UNIX)
\amt\samples (Windows)
```

Overriding the default location and names

You can override where the AMI looks for the repository and local host files by using an environment variable:

```
ADDENVVAR ENVVAR(AMT_DATA_PATH) VALUE('/directory') (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')
ADDENVVAR ENVVAR(AMT_HOST) VALUE('myHostFile.xml')

export AMT_REPOSITORY = myData.xml
export AMT_HOST = myHostFile.xml

set AMT_REPOSITORY = myData.xml
set AMT_REPOSITORY = myData.xml
set AMT_REPOSITORY = myData.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:

```
setRepositorv(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.

```
<?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 = "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:

```
= "QMNAME1"
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.

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 (or an LDAP directory service) provides definitions for policies and services. If you do not use a repository file (or an LDAP directory service), AMI uses its built-in definitions. For more information, see "Chapter 19. Defining services, policies, and policy handlers" on page 491.

For information about LDAP support, see "Chapter 20. Lightweight Directory Access Protocol support" on page 505.

Local host and repository files (OS/390)

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.

Local host and repository files (OS/390)

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:

```
connectionName1
                = "OMNAME1"
```

The repository connection names are not limited to the values shown (connectionName1 and connectionName2). Any name can be used provided it is unique in both the repository and local host files, and consistent between the two.

"Repository and local host caches" on page 475 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 19. Defining services, policies, and policy handlers" on page 491.

"Repository and local host caches" on page 475 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.

Building a cache from source xml files: Specify the input xml repository data file with a DD statement for the file AMT. Specify the input host data file with a DD statement for the file AMTHOST.

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

```
//GO EXEC PGM=AMTASM10
//STEPLIB DD DSN=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 that the cache generator returns are described in "Cache generator and viewing the repository messages" on page 476.

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 473 and "CICS" on page 473.

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.

Local host and repository files (OS/390)

Cache generator and viewing the repository 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 */
/* CICS: Ensure that one of the language-specific datasets is */
/* on 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:
"AMT0005I AMI host file cache created"
/* A host file cache was successfully created.
/* User Response:
/* None.
```

The AMI Administration Tool

The AMI Administration Tool is for use on Windows NT Version 4 and Windows 2000 only.

Installation

The AMI Administration Tool is packaged with the AMI in ma0f_nt.zip and optionally installed with the AMI using the setup InstallShield program (see "Installation on Windows" on page 466). It is installed in sub-directory amt\AMITool.

To start the AMI Administration Tool, 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\samples directory, and open the sample repository file amt.xml. You should see a number of services, policies, and policy handlers in the navigation pane on the left. Select one of them by clicking on it, and you should see its attributes displayed in the pane on the right.

Operation

The AMI Administration Tool enables you to create definitions for:

Service points used to create sender or receiver services

Distribution lists must include at least one sender service

Publishers must include a sender service as the broker service

Subscribers must include sender and receiver services as the

broker and receiver services

Policies contain sets of attributes: initialization, general,

send, receive, publish, subscribe, handler

Policy handlers used to create policy handler library definitions

The default attributes provided by the tool are as specified in "Service definitions" on page 494, "Policy definitions" on page 497, and "Policy handler definitions" on page 504.

When you have entered the definitions you require, select **Save** in the **File** menu to save them as an XML-format repository file. It is recommended that you define all your services, policies, and policy handlers in the same repository file.

The repository file must be copied to a location where it can be accessed by the AMI (see "Local host and repository files (AS/400, UNIX®, and Windows)" on page 471). If the Application Messaging Interface is on the same system as the tool, the repository file can be copied to the AMI directory. Otherwise, the repository file must be transferred to that system using a method such as file sharing or FTP.

Notes:

- 1. To open an existing repository file (including the amt.xml file provided in the samples directory), the repository file and the amt.dtd file must both be in the same directory.
- 2. If you are using the AMI Administration Tool to create an XML repository file for an earlier version of the AMI (for example, using the AMI 1.2

The AMI Administration Tool

Administration Tool to create an XML file for use with AMI 1.1), then you must ensure that you include the newer amt.dtd file with the new XML file in the same directory, replacing the old amt.dtd file as necessary.

For further information, refer to the AMI Administration Tool online help, or "Chapter 19. Defining services, policies, and policy handlers" on page 491 and "Updating LDAP from a repository" on page 509.

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 497).

Using MQSeries Integrator Version 1

If you are using the AMI with MQSeries Integrator Version 1, the Service Type for the sender service point must be defined in the repository as MQSeries Integrator V1 (see "Service definitions" on page 494). This causes an MQRFH header containing application group and message type name/value elements to be added to a message when it is sent.

The Application Group definition is included in the policy send attributes (see "Policy definitions" on page 497). The message type is defined as the message format value set in the message object (using amMsgSetFormat, for example). If this is set to AMFMT_NONE, the message type is defined as the Default Format for the sender service point (a maximum of eight characters in 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 494). 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 MQSeries Integrator Version 1 (MQSI 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.

Connecting to MQSeries

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 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 recognize 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 local queue manager by typing the following on the command line, where QMName is the name of your MQSeries queue manager: STRMQM MQMNAME(QMName)

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

For 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.

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

We recommend that you run one or more of the sample programs to verify that you have installed the Application Messaging Interface correctly.

For the OS/390 platform, see "Sample programs for OS/390" on page 484.

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	SendForget	SendForget
A sample that receives a message, with no selection.	amtshrcv	amtsorcv	Receiver	Receiver
A sample that sends a request and receives a reply to this request (a simple client program).	amtshclt	amtsoclt	Client	Client
A sample that receives requests and sends replies to these requests (a simple server program).	amtshsvr	amtsosvr	Server	Server
A sample that periodically publishes information on the weather.	amtshpub	amtsopub	Publisher	Publisher
A sample that subscribes to information on the weather, and receives publications based on this subscription.	amtshsub	amtsosub	Subscriber	Subscriber
A sample that sends messages using simulated group support.	-	amtsosgs	-	-
A sample that receives messages using simulated group support.	-	amtsosgr	-	-
A sample that performs a file transfer send on a user supplied text file.	amtshfsn	amtsofsn	SendFile	SendFile
A sample that performs a file transfer receive on a user supplied text file.	amtshfrc	amtsofrc	RcvFile	RcvFile
A sample policy handler library that inserts a CICS (MQCIH) or IMS (MQIIH) header at the start of each message that is sent, or that performs archiving by sending each message that is sent to a separate archive queue.	-	amtsphlr (UNIX and Windows only)	-	-

To find the source code and the executables for the samples, see "Directory structure" on page 445 (AIX), page 449 (AS/400), page 455 (HP-UX), page 463 (Solaris), and page 468 (Windows).

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 local queue manager by typing the following on the command line:

```
STRMOM MQMNAME (OMName)
```

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</pre>
                                                   (UNIX)
runmqsc {QMName} < {Location}\amtsamp.tst</pre>
                                                   (Windows)
```

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 471).

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:

```
strmgbrk -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 the name of your MQSeries Integrator Version 2 broker:

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

- 2. Use the Control Centre to create a simple message flow. This should consist of an MQInput node with the Output terminal connected to the input terminal of a Publication node.
- 3. In the properties for the MQInput node, select the "Basic Properties" pane and set the Queue Name to:

```
SYSTEM.BROKER.DEFAULT.STREAM
```

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.
- 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 445 (AIX), page (AS/400), page 454 (HP-UX), page 462 (Solaris), and page 467 (Windows).

Running the C and C++ samples

You can run a C or C++ sample program by typing the name of its executable at a command line. For example:

```
amtsosnd
```

will run the "Send and forget" sample written using the C object interface.

Running the Java samples

The AMI samples for Java are in a package called:

```
com.ibm.mq.amt.samples
```

To invoke them, you need to specify the name of the sample plus its package name. For example, to run the "Send and forget" sample, use:

```
java com.ibm.mq.amt.samples.SendForget
```

Running the policy handler library sample

The sample policy handler library (AMT.SAMPLE.HANDLER) archives each message when it is sent using either a sender or distribution list, by putting a copy of the message to an additional MQSeries queue

(AMT.SAMPLE.POLICY.HANDLER.QUEUE) on the local queue manager.

To run the policy handler, the MQSeries queue AMT.SAMPLE.POLICY.HANDLER.QUEUE must exist. To create this queue, you

can run the amtsamp.tst script file when prompted during installation, or you can run the sample MQSC script after installation (see "MQSeries objects" on page 482).

Also, the policy handler must be defined in the repository and referenced in the policy specified with the send function by an application. The sample repository file (amt.xml) that is provided with the AMI includes a policy handler definition named AMT.SAMPLE.HANDLER for this library. It also includes a policy definition named AMT.SAMPLE.POLHDLR.POLICY that includes a reference to AMT.SAMPLE.HANDLER. See "Repository and host files" on page 482.

You can use the policy AMT.SAMPLE.POLHDLR.POLICY to exercise the sample policy handler by recompiling amtshclt.c or amtsoclt.c with the AMT_RUN_HANDLERS preprocessor directive specified (that is, using the -D option with the cl command). This causes the AMT.SAMPLE.POLHDLR.POLICY policy name to be used instead of the one that is normally used.

For a more detailed description, see "The AMI policy handler sample program (amtsphlr)" on page 487.

Running the sample programs (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

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

Table 6. The sample programs for OS/390 ('batch' includes RRS-batch) (continued)

Description	C High level	C Object level	COBOL High level	COBOL Object level
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 459.

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 257.

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:

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

Repository and host files

The sample repository AMT (located in hlq.SCSQPROC) is appropriate for use with all the sample programs (though many of the samples will work correctly without a repository). If you wish to use the repository file, ensure that the sample program has access to it, as described in "Local host and repository files (OS/390)" on page 473.

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 473.

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:

```
strmgbrk -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:

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

```
mqsistart {BrokerName}
mgsistart 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 Oueue 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 it.

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 458.

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.

Each method resolves to userId.MYTEST.FILE.

Running the batch samples

You can run batch sample programs by constructing a piece of JCL to run the program, and submitting that JCL from ISPF. The batch samples can also be used as RRS-batch sample programs.

Running the CICS samples

Ensure that the CICS DFHRPL includes the load library containing the sample, as well as the AMI library. Define the sample program to CICS, as well as a transaction to run the program. Finally, ensure that the AMI library, sample program and sample transaction are installed in your CICS region. Type the transaction name into a CICS console to run the sample.

Running the IMS samples

Ensure that the IMS message processing region JCL includes the load library that contains the sample, as well as the AMI library. Define the sample program and transaction name to IMS. Type the transaction name into an IMS console to run the sample.

The AMI policy handler sample program (amtsphlr)

The AMI Policy Handler sample program, amtsphlr, is implemented using the following files:

- amtsphlh.h: C header file for policy handler sample
- amtsphlr.c : C source for policy handler sample
- amtsphlr.exp: AIX and HP-UX C export file for policy handler sample
- amtsphlr.def: Windows C definition file for policy handler sample

For AS/400, the AMI Policy Handler sample program is implemented using the following files:

- amtsphlh in QMQMAMI/H : C header file for policy handler sample
- amtsphlr in QMQMAMI/QCSRC : C source for policy handler sample
- amtsphlr in QMQMAMI/QSRVSRC : AIX and HP-UX C export file for policy handler sample

This sample program illustrates the following policy handler operations:

 Creation. The creation of a policy handler context and the return of a context handle.

• Initialization. The initialization of the policy handler and invocation point registration.

The sample policy handler registers functions for the following invocation points:

AMINV PRE MQCONNX AMINV PRE MQOPEN AMINV PRE MQPUT

- Deletion. The deletion of the policy handler context.
- Invocation. The invocation of a policy handler to perform a simple invocation print function.
- Auditing. The invocation of a policy handler with policy initialization parameters used as a directive to log each message on a separate audit queue when it is sent.
- Header insertion. The invocation of a policy handler with a service custom parameter used as directive to insert an additional header at the start of the message. This sample program illustrates MQSeries CICS (MQCIH) and IMS (MQIIH) header insertion, with policy invocation parameters used to provide the CICS or IMS program name.

These functions illustrate the use of the amLibTraceText AMI callback function to perform tracing when requested.

A definition for the sample policy handler is provided in the sample repository file amt.xml that is provided with AMI. This specifies the policy handler AMT.SAMPLE.HANDLER with library name amtsphlr. The corresponding library file must be located in the AMI handlers directory for the platform concerned. The name of the corresponding library file requires the prefix 1ib on AIX, HP-UX and Sun Solaris, and requires the appropriate suffix for the platform concerned:

- AIX (non-threaded) .a
- AIX (threaded) r.a
- _r AS/400 (threaded)
- HP-UX r.sl
- Sun Solaris . SO
- .dl1 Windows

The policy handler name AMT.SAMPLE.HANDLER is also specified in the list of handlers for the policy AMT.SAMPLE.POLHDLR.POLICY. To invoke this policy handler, an application must create a policy object with the name AMT.SAMPLE.POLHDLR.POLICY and specify this as a parameter on an AMI open or send function. To invoke this policy handler using the amtsoclt.c or amtshclt.c sample programs, you must recompile this program using the -D AMT_RUN_HANDLERS compiler option (to define the symbol AMT_RUN_HANDLERS).

For details about how to run the sample policy handler, see "Running the policy handler library sample" on page 483.

Required entry points

The sample policy handler library implements and exports the three required entry points as follows:

amPhlCreate

This is called on creation of the first policy to include this policy handler in

its list of handlers. It allocates the memory required for its own instance data, and returns a context handle that will be passed back to the policy handler on each subsequent call.

The context handle enables the policy handler to manage and access its own memory context information across invocations. The value returned for the context handle must be non-NULL. A policy handler context has session scope, that is a separate **amPhlCreate** for the same policy handler can occur for each AMI session.

The context handle can itself be used to hold the context pointer if its size is suitable. Otherwise, on platforms where this is not possible, the pointer must be stored in global memory. Note that when the pointer is stored in global memory, if multiple sessions reference the same policy handler, the context handle can serve as an index to a pointer table in global memory. However, this is not illustrated in this sample program.

amPhlInitialize

This is called immediately after **amPhlCreate** to enable the policy handler to call back into the AMI and register the set of invocation points for which it wishes to be called and specify their function address values. This sample program registers the following functions and invocation points:

Function Name	Invocation point
InvocationPrint	AMINV_PRE_MQCONNX
InvocationPrint	AMINV_PRE_MQOPEN
AddHeader	AMINV_PRE_MQPUT
Audit	AMINV_POST_MQPUT

amPhlDelete

This is called when the session is deleted to enable the policy handler to free any memory it allocated during **amPhlCreate** processing.

Invocation point functions

The sample policy handler also implements the following functions that it registers with the required invocation points during **amPhlInitialize** processing, as listed earlier. These implement the real function provided by the policy handler.

InvocationPrint

This function prints a message that identifies the invocation point (that is, in this example, either AMINV_PRE_MQCONNX or AMINV_PRE_MQOPEN).

Audit This function is invoked after a successful MQPUT. If the handler initialization parameters specified AUDIT, this function logs a copy of the same message to a separate MQSeries audit queue, that is:

AMT.SAMPLE.POLICY.HANDLER.QUEUE on the local queue manager to which the AMI session is connected.

AddHeader

This function is invoked immediately before an MQPUT.

If the custom parameters for the sender service specify CICS, this function constructs an MQSeries MQCIH (CICS) message header and uses the AMI call back function **amxMsgAssemble** to add this at the start of the message. The policy invocation parameters are used to specify the CICS program name.

If the custom parameters for the sender service specify IMS, this function constructs an MQSeries MQIIH (IMS) message header and this uses the AMI call back function amxMsgAssemble to add this at the start of the message. The policy invocation parameters are used to specify the IMS program name.

The function returns AMPH_CONTINUE in the control flags, which causes the AMI to issue the MQPUT to send the modified message.

If you use a policy handler to add a CICS header, this enables your AMI application to run CICS Distributed Program Link (DPL) programs on a CICS Transaction Server using MQSeries and the MQSeries-CICS bridge as the communication mechanism.

If you use a policy handler to add an IMS header, this enables your AMI application to run IMS/ESA transactions on an IMS Transaction Server using MQSeries and the MQSeries-IMS bridge as the communication mechanism.

Chapter 19. Defining services, policies, and policy handlers

Definitions of services, policies, and policy handlers created by a system administrator are held in a *repository*. The Application Messaging Interface provides a tool to enable the administrator to set up new services, policies, and policy handlers, and to specify their attributes (see "The AMI Administration Tool" on page 477).

This chapter contains:

- "Services, policies, and policy handlers"
- "Service definitions" on page 494
- "Policy definitions" on page 497
- "Policy handler definitions" on page 504

Services, policies, and policy handlers

A repository file contains definitions for *policies*, *policy handlers*, and *services*. A service is the generic name for any object to which a send or receive request can be issued, that is:

- Sender
- Receiver
- Distribution list
- Subscriber
- Publisher

Sender and receiver definitions are represented in the repository by a single definition called a *service point*.

Policies, policy handlers, and services other than distribution lists, can be created with or without a corresponding repository definition; distribution lists can be created only with a corresponding repository definition.

To create a service, policy, or policy handler using the repository, the repository must contain a definition of the appropriate type with a name that matches the name specified by the application. To create a sender object named 'DEBITS' (using amSesCreateSender in C, for example) the repository must have a service point definition named 'DEBITS'.

Policies, policy handlers, and services created with a repository have their contents initialized from the named repository definition.

If the repository does not contain a matching name, a warning is issued (such as AMRC_POLICY_NOT_IN_REPOS). The service or policy is then created without using the repository (unless it is a distribution list).

Policies and services created without a repository (either for the previous reason, or because the repository is not used), have their contents initialized from one of the system provided definitions (see "System provided definitions" on page 492).

Definition names in the repository must not start with the characters 'AMT' or 'SYSTEM'.

Services, policies, and policy handlers

System provided definitions

The AMI provides a set of definitions for creating services, policies, and policy handlers without reference to a repository.

Table 7. System provided definitions

Definition	Description
AMT.SYSTEM.POLICY	This provides a policy definition with the defaults specified in "Policy definitions" on page 497, except that Wait Interval Read Only is not selected in the Receive attributes.
AMT.SYSTEM.SYNCPOINT.POLICY	This provides a policy definition the same as AMT.SYSTEM.POLICY, except that Syncpoint is selected in the General attributes.
AMT.SYSTEM.SENDER	This provides a sender definition with the defaults specified in "Service definitions" on page 494, with the Queue Name the same as the Sender object.
AMT.SYSTEM.RESPONSE.SENDER	This provides a sender definition the same as AMT.SYSTEM.SENDER, except that Definition Type, Queue Name and Queue Manager Name are set to 'Undefined' (that is, set when used).
AMT.SYSTEM.RECEIVER	This provides a receiver definition the same as AMT.SYSTEM.SENDER.
AMT.SYSTEM.SUBSCRIBER	This provides a subscriber definition in which the Sender Service has the same name as the Subscriber object, and the Receiver Service has the same name with the suffix '.RECEIVER'.
AMT.SYSTEM.PUBLISHER	This provides a publisher definition in which the Broker Service has the same name as the Publisher object.

System default objects

A set of system default objects is created at session creation time. This removes the overhead of creating the objects from applications using these defaults. The system default objects are available for use from the high-level and object-level interfaces in C. They cannot be accessed using C++ or Java (these languages can use the built-in definitions to create an equivalent set of objects if required).

The default objects are created using the system provided definitions, as shown in the following table.

Table 8. System default objects

Default object	Definition
SYSTEM.DEFAULT.POLICY	AMT.SYSTEM.POLICY
SYSTEM.DEFAULT.SYNCPOINT.POLICY	AMT.SYSTEM.SYNCPOINT.POLICY
SYSTEM.DEFAULT.SENDER	AMT.SYSTEM.SENDER
SYSTEM.DEFAULT.RESPONSE.SENDER	AMT.SYSTEM.RESPONSE.SENDER
SYSTEM.DEFAULT.RECEIVER	AMT.SYSTEM.RECEIVER
SYSTEM.DEFAULT.SUBSCRIBER	AMT.SYSTEM.SUBSCRIBER
SYSTEM.DEFAULT.PUBLISHER	AMT.SYSTEM.PUBLISHER

Services, policies, and policy handlers

Table 8. System default objects (continued)

Default object	Definition
SYSTEM.DEFAULT.SEND.MESSAGE	N/A
SYSTEM.DEFAULT.RECEIVE.MESSAGE	N/A

The default objects can be used explicitly using the AMI constants (see "Appendix B. Constants and structures" on page 561), or used to provide defaults if a particular parameter is omitted (by specifying NULL, for instance).

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

Service definitions

This section gives the service definitions for:

- service point (sender/receiver)
- distribution list
- subscriber
- publisher

Service point (sender/receiver)

Table 9. Service point (sender/receiver)

Attribute	Comments
Name	Mandatory name, specified on AMI calls. 1
Queue Name	Mandatory name of the queue representing the service that messages are sent to or received from. 2
Queue Manager Name	Name of the queue manager that owns Queue Name. If blank, the local queue manager name is used. 2
Model Queue Name	Name of a model queue definition used to create a dynamic queue (normally a Reply Service to receive response messages). Required if the Definition Type is 'Dynamic'.
Dynamic Queue Prefix	Name of a prefix used when creating a dynamic queue from Model Queue Name. Required if the Definition Type is 'Dynamic'. If the last non-blank character in positions 1 to 33 of the prefix is '*', the '*' is replaced by a string that guarantees that the name generated is unique. 2
Definition Type	Defines how the AMI obtains the queue name for the service point. If set to 'Predefined' (the default), the Queue Name and Queue Manager Name as specified earlier are used. If set to 'Dynamic', the Model Queue Name and Dynamic Queue Prefix are used to create a dynamic queue.
Service Type	Defines the RF header (if any) that is sent with the message data, and the parameters within the header.
	Set to 'Native' for a native MQSeries 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).
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 4
Default MCD Format	Defines the default message format value. This is added to any message being sent using this service point if the Service Type is '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.)
Custom Parameters	Free-format text parameter string that is associated with the service point and passed to the policy handler library on each invocation to provide service-specific information. The implementor of the policy handler defines the meaning and syntax of the string. The string is specific to the policy handler library, and the policy handler library parses and interprets the string.
	A typical use of this field might be to further customize the service-type as CICS or IMS, where the policy handler library is responsible for inserting and removing a MQCIH or MQIIH header at the start of the message.

Notes:

- 1 The name is a maximum of 256 characters, and can contain the following characters: A-Z, a-z, 0-9, '.', '/', __' and '%'.
- 2 The name is a maximum of 48 characters, and can contain the following characters: A-Z, a-z, 0-9, '.', '/',
- 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.
Notes	

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, θ -9, '.', '/', and '%'.

Policy definitions

This section describes the policy definitions for the following attributes:

- initialization
- general
- send
- receive
- subscribe
- publish
- handler

Initialization attributes

Table 13. Initialization attributes

Attribute	Comments		
Name	Mandatory policy name, specified on AMI calls. 1		
Connection Name	If Connection Mode is set to 'Real', Connection Name is the name of the queue manager the application will connect to. If blank, the default local queue manager is used. If Connection Mode is 'Logical', the Connection Name attribute is required and is the name the logical connection used with the local host file to generate the queue manager to whice 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.		
Client Channel Name	For an MQSeries client connection, the name of the server-connection channel. Can be used instead of the MQSERVER environment variable on the MQSeries client with the TCP/IP transport type.		
Client TCP Server Address	For an MQSeries client connection, the TCP/IP host name (and optional port) of the MQSeries server. Can be used instead of the MQSERVER environment variable on the MQSeries client with the TCP/IP transport type.		

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, θ -9, '.', '/', '_' and '%'.

3 The Connection Type that is established on the first session open persists for the entire process. You cannot change this by opening a subsequent session using a policy with a different Connection Type.

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 Transport	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.
New Correl Id	Selected Not selected	Not selected	When selected, each message is sent with a new Correl Id (except for response messages, where this is set to the Message Id or Correl Id of the request message).
Response Correl Id	Message Id Correl Id	Message Id	The Id set in the Correl Id of a response or report message. This is set to either the Message Id or the Correl Id of the request message, as specified.
Exception Action	Discard DLQ	DLQ	Action when a message cannot be delivered. When set to DLQ, the message is sent to the dead-letter queue. When set to Discard, it is discarded.
Report Data	Report With Data With Full Data	Report	The amount of data included in a report message, where Report specifies no data, With Data specifies the first 100 bytes, and With Full Data specifies all data.

Table 15. Send attributes (continued)

Attribute	Values	Default	Comments
Report Type Exception	Selected Not selected	Not selected	When selected, Exception reports are required.
Report Type COA	Selected Not selected	Not selected	When selected, Confirm on Arrival reports are required.
Report Type COD	Selected Not selected	Not selected	When selected, Confirm on Delivery reports are required.
Report Type Expiry	Selected Not selected	Not selected	When selected, Expiry reports are required.
Segmentation	Selected Not selected	Not selected	When selected, Segmentation of the message is allowed.
Split File	Logical Physical	Physical	When set to Logical, the file is split into separate messages at record boundaries, as determined by the value of File Record Length. On Windows, HP-UX, AIX, and Sun Solaris, if the File Record Length is zero, this is the end of a line. On OS/390, this is a record boundary.
			When set to Physical, the file is split into separate messages on boundaries that are determined by AMI.
File Record Length	0-99999999	0	When Split File is set to Logical, a non-zero value specifies the boundary to use to split a file into individual messages.
			When Split File is set to Physical, this value is ignored.
Bind On Open	Yes No As Transport	As Transport	Bind On Open controls the binding of a service point to a particular instance of 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
Implicit Open	Selected Not selected	Selected	When selected, the queue is opened implicitly (must be selected for the C and COBOL high-level interfaces).
Leave Queue Open	Selected Not selected	Selected	When selected, a queue that was implicitly opened will be left open.
Delete On Close	Yes No Purge	No	When set to Yes, temporary dynamic queues, and permanent dynamic queues that contain no messages, are deleted when closed. When set to No, dynamic queues are not deleted when closed. When set to Purge, dynamic queues are deleted
			when closed, even if the queues contain messages.
Wait Interval	0-99999999 Unlimited	Unlimited	A period of time (in milliseconds) that the receive waits for a message to be available.
Wait Interval Read Only	Selected Not selected	Selected	When selected, an application cannot override the Wait Interval value in the policy object.
Convert	Selected Not selected	Selected	When selected, the message is code page converted by the message transport when received.
Wait For Whole Group	Selected Not selected	Selected	When selected, all messages in a group must be available before any message is returned by the receive.
			When not selected, AMRC_NO_MSG_AVAILABLE may be returned to the application before the complete group is received. In this case, any simulated group state information is destroyed and any remaining messages in a simulated group are orphaned.
Handle Poison Message	Selected Not selected	Selected	When selected, poison message handling is enabled.
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 handles 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.

In all cases, a warning is returned and the message is returned to the application (even if it was successfully queued on the backout-requeue queue). Also, the message does not disappear from the original queue from where it is received, unless the application explicitly performs a commit.

Policy definitions

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.

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.

Handler attributes

Table 19. Handler attributes

Attribute	Comments	
Handler	The name that is assigned to this policy handler when it is created.	
Invocation Parameters	A free-format text parameter string that is associated with the policy and passed to the policy handler library on each invocation to provide policy-specific information. The implementor of the policy handler defines the meaning and syntax of the string. The string is specific to the policy handler library, and the policy handler library parses and interprets this string. A typical use of this field is to pass policy-specific options to the policy handler library.	

Policy handler definitions

This section describes the attributes for a policy handler library definition.

Policy handler attributes

Table 20. Policy Handler attributes

Comments				
Mandatory policy handler name. Used to identify this policy handler on the list of handlers that are associated with a policy.				
The name of the policy handler library file, excluding any directory information, prefix, and platform-specific extension, that the AMI should load for this policy handler. For AIX, HP-UX, and Solaris, the AMI prefixes 1 ib to the specified name. The AMI appends				
the platform-dependent file extension to the specified name as follows:				
.a AIX (non-threaded)				
_r.a AIX (threaded)				
_r AS/400 (threaded)				
_r.sl HP-UX				
.so Solaris				
.dll Windows and OS/390				
For AS/400 non-threaded, there is no file extension.				
A free-format text parameter string that is passed to the policy handler library on initialization. The implementor of the policy handler defines the meaning and syntax of the string. The string is specific to the policy handler library, and the policy handler library parses and interprets this string.				

Notes:

1 The name is a maximum of 256 characters, and can contain the following characters: A-Z, a-z, 0-9, '.', '/', and '%' ' and '%'.

Chapter 20. Lightweight Directory Access Protocol support

Information that describes the various users, applications, and other resources that are available on a computer network can be collected into a specialized database or repository called a directory.

Directories are usually accessed using a client/server computing model:

- Applications that need to access and update information in the directory issue requests to a directory server.
- The server manages the storage and retrieval of data in the directory.

The Lightweight Directory Access Protocol (LDAP) is an open industry standard that defines a protocol for the requests and responses that flow between directory clients and servers. AMI support for LDAP means that service, policy, and policy handler definitions can be stored and accessed across networks by using a directory as an alternative to distributing repository files.

It is possible to update local or remote LDAP directories with AMI information from a repository file. On Windows NT and Windows 2000, you can use the AMI Administration Tool to do this.

Environment variables are used to configure AMI applications to retrieve AMI information from the directory. A directory schema defines the objects and attributes that may be stored in and retrieved from a directory. The AMI includes schema definitions that provide object classes and attributes for AMI service (service point, distribution list, publisher, subscriber), policy, and policy handler objects.

AMI Version 1.2 supports LDAP on all AMI platforms. OS/390 and AS/400 operating systems include LDAP client software that is used by AMI. On the other platforms, the IBM SecureWay Directory Version 3.2 Client (or later) must be installed. The schema definition files and information update facilities that are provided with the AMI are supported on the following LDAP directory servers:

- LDAP V3 IBM SecureWay Directory
- Microsoft Active Directory

These are described in the following sections.

Getting Started With LDAP

If your directory server does not have the AMI LDAP schema installed, install this first. See "Installation" on page 506. Note that you do not need to install the AMI on the machine that hosts the directory server.

To use the AMI with an LDAP directory:

- Create your AMI repository xml file in the normal way using the AMI Administration Tool.
- 2. Update your directory server with the AMI repository information. You can do this directly from the AMI Administration Tool on Windows, or on any platform by using the xml repository file with the amtldup command line program. See "Updating LDAP from a repository" on page 509.

LDAP support

To access AMI repository information in a SecureWay or Active Directory when using your AMI application, you must set the AMT_REPOSITORY environment variable to an LDAP URL. See "Directory search" on page 512. Alternatively, in C++ or Java, you can use the **AmSessionFactory setRepository** method to specify the LDAP URL. See "setRepository" on page 199 (C++) or "setRepository" on page 401 (Java).

SecureWay Directory

SecureWay Directory is a directory service that is based on LDAP and DB2. SecureWay Directory at LDAP V3 level is available on AIX, AS/400 (Version 4 Release 5), OS/390 (Version 2 Release 9 or later), Sun Solaris, and Windows NT platforms.

The SecureWay Directory supports the Simple Authentication and Security Layer (SASL) by using Kerberos or the CRAM-MD5 mechanism for authentication. The Secure Sockets Layer (SSL) or Transport Layer Security (TLS) can be used for transport security and authentication.

The AMI includes schema definition files for all the SecureWay Directory platforms.

Active Directory

The Active Directory is a directory service that is included with the Windows 2000 Server.

The Active Directory service provides an interface that supports LDAP V3. There is also a Component Object Model (COM) based interface, Active Directory Service Interfaces (ADSI). ADSI enables client applications to communicate with any directory services that are compliant with LDAP or Novell Directory Services (NDS). Active Directory is available only on Windows 2000 domain controllers.

The Active Directory provides SASL negotiation, using Kerberos as the default mechanism. SSL can be used for encryption.

The AMI includes a schema definition file to extend the Active Directory schema for AMI use.

Installation

For the AMI platforms, the SecureWay Directory Version 3.2 Client (or later) is available as a free download from the IBM Web site, on:

http://www.ibm.com/software/network/directory

For each computer where you wish to use the AMI LDAP facilities, the SecureWay Client must be installed and available. This includes the computer from where you install the schema, and the computer from where you update the directory.

For the directory server platforms, there are schema extension files and scripts to install the schema. These files and scripts vary according to the directory service, not the operating system. Generally, each directory server that will be used to hold AMI information must have the AMI schema extensions installed once, usually by an administrator of the directory. For a Windows 2000 forest, the AMI schema extensions are installed once to extend the Active Directory schema, then they will be replicated throughout the forest.

If you use a version of the SecureWay Directory that is later than Version 3.2.1, the AMI schema is installed as part of the base schema.

Note that the target directory server itself must be configured with a suffix and initial data before AMI data can be stored in, or retrieved from, the directory.

SecureWay Directory

The AMI includes files that contain the AMI schema classes and attributes. If the AMI schema is not installed as part of the base schema (which depends on the version of the SecureWay Directory Server installed), you can use these files to install the AMI schema to the directory server.

Note: If the directory schema is held under a suffix other than cn=schema (the default), you must modify the schema file to reference the appropriate

To install the AMI schema to a directory server on AIX, AS/400, OS/390 TDBM, Solaris or Windows, you run the ldapmodify program, which is provided with SecureWay Directory Client software. To install the AMI schema for OS/390 RDBM installations of the SecureWay LDAP server, see "OS/390 RDBM" on page 508.

Note: OS/390 Version 2 Release 10 and later supports both TDBM and RDBM. However, OS/390 Version 2 Release 9 supports only RDBM (DB2 backend).

To install the AMI schema using the ldapmodify program, use the following steps:

- 1. Change directory to the AMI ldap directory (see "Directory structure" on page 445 (AIX), page 455 (HP-UX), page 463 (Solaris), or page 468 (Windows).
- 2. Issue the ldapmodify command:

ldapmodify -h host -p port -D dn -w password -f filename

where:

host The TCP address of the computer on which the directory server

is running (optional). The default value is localhost.

port The port that the directory server uses (optional). The default

value is 389.

dn The distinguished name (DN) of a directory administrator. This

name is used to bind to the directory.

The password for the distinguished name of the directory password

administrator.

filename The file that contains the definitions AMI schema classes and

attributes.

For directory servers that run on AIX, AS/400, Solaris, or

Windows, this is amtsw.ldif.

For OS/390 TDBM servers, this is amtsw390.ldif.

If the directory server is configured to require high security, you can use further ldapmodify options. For further details about the ldapmodify command, refer to the SecureWay Directory documentation.

OS/390 RDBM

This section describes how to install the AMI schema for OS/390 RDBM installations of the SecureWay LDAP server.

For OS/390 RDBM, at installation, two schema configuration files are created in the AMI samples directory (see "Directory structure" on page 445 (AIX), page 449 (AS/400), page 455 (HP-UX), page 463 (Solaris), or page 468 (Windows). These files are:

- amtsw390.at, which defines the AMI LDAP attributes
- amtsw390.oc, which defines the AMI LDAP object classes

To extend the directory, the administrator must:

- 1. Stop the directory server.
- 2. Transfer the RDBM schema definition files as text to the OS/390 Hierarchical File System, that is:
 - amtsw390.at to /etc/ldap/amtsw390.at
 - amtsw390.oc to /etc/ldap/amtsw390.oc
- 3. Edit the Secureway LDAP server configuration file /etc/ldap/slapd.conf to add the following two lines:

```
include /etc/ldap/amtsw390.oc
include /etc/ldap/amtsw390.at
```

4. Restart the Secureway LDAP server.

Active Directory

Installation includes a script, admqami.vbs, and the AMI schema classes and attributes in the file admqami.ldf. To extend the directory schema, you must log on to an appropriate Windows 2000 domain as a domain administrator, then run the supplied script.

Note: Active Directory schema changes are generally permanent and cannot be removed.

Changes to the Active Directory schema are replicated throughout the domain forest, so usually, you need to perform the schema update only once for each enterprise.

To extend the directory schema, use the following steps:

- 1. Ensure that the current Schema Master for the Active Directory is online. You can find and change the Schema Master by using the Active Directory Schema MMC snap-in. This is provided with Windows 2000 Server and Windows 2000 Advanced Server. For details, refer to the Microsoft documentation about the Active Directory. The snap-in also indicates whether the schema master is available.
- 2. On the domain controller where you wish to initiate the schema update (this does not need to be the Schema Master), ensure that the following registry flag is set to 1:

```
HKEY LOCAL MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\
Parameters\Schema Update Allowed
```

To do this, either use the Active Directory Schema MMC snap-in, or use the regedit command.

This setting means that the schema can be modified from the selected domain controller, or from a workstation in the domain.

- 3. Log on as a domain administrator to a computer on which AMI is installed. This computer must be a Windows 2000 domain controller or Windows 2000 Professional computer.
- 4. At a command prompt, change directory to the AMI ldap directory (see "Directory structure (Windows)" on page 468).
- 5. Run the following script file:

amtad.vbs

This script determines the current Schema Master and runs the Microsoft LDIFDE utility to add definitions from the admqami.ldf file to the Active Directory schema. When prompted, select **Yes** to confirm that you wish to update the schema.

If the update fails, the script displays the error code that the Microsoft LDIFDE utility returns.

Uninstallation

If you uninstall MQSeries, there are no special considerations or actions required for LDAP. Entries that are created in target directories are not removed. If required, you can delete schema extensions from the SecureWay Directory manually. However, for the Active Directory, you can only disable the schema extensions.

Updating LDAP from a repository

It is possible to update local or remote LDAP directories with AMI information from a repository file. On Windows NT and Windows 2000, you can use the AMI Administration Tool to do this. From the AMI Administration Tool, you can update a remote LDAP directory on any supported platform, for example, OS/390 or Solaris.

On AIX, HP-UX, Sun Solaris, and Windows, you can update an LDAP directory using the command line.

Using the AMI Administration Tool

To use the AMI Administration Tool to save the current repository file to a LDAP directory:

- 1. Start the AMI Administration Tool (see "The AMI Administration Tool" on page 477).
- 2. Select File→Update LDAP Directory.

If the information is not saved to a file, it is automatically saved, or you are prompted to save it to a new file.

3. When the Update LDAP Directory window is displayed, enter the following information:

LDAP Server Name The TCP address of the LDAP server host

(default, localhost).

Port The port to connect to the LDAP server (default

389).

Base Dn The distinguished name under which the AMI

information is placed in the Directory

LDAP support

Information Tree. The default is the container cn=AMIInfo under the default directory naming

context.

Authentication Type The method to use to log in to the directory.

Select from:

None

• Simple (the default)

CRAM MD5

Kerberos

External

Bind dn The user Id. This user must have permission to

create entries in the target directory.

If the Authentication Type is set to None, Kerberos, or External, you can leave this field

blank.

Password Password for the user Id.

> If the Authentication Type is set to None, Kerberos, or External, you can leave this field

Use SSL Select this option if Secure Sockets Layer (SSL)

is used.

Key File Name The file that contains the encryption keys to use

with SSL. Optionally, use the Browse button to

select the file.

If you do not select Use SSL, you can leave this

field blank.

Key File Password The password for the key file.

If you do not select Use SSL, you can leave this

field blank.

Private Key Name The private key name to use in the key file.

If you do not select Use SSL, you can leave this

field blank.

LDAP Version The version of the LDAP server (2 or 3).

4. When your entries are complete, click **OK**. The AMI LDAP update program runs and updates the specified directory with information from the repository

When the operation completes successfully, a success message is displayed. Otherwise, a message that includes the error code is displayed. For details about any error messages, see "LDAP error codes" on page 556, or the AMI Administration Tool online help.

Using the command line

The AMI LDAP Directory Update Program, amtldup, is provided so that you can update a directory from a repository xml file using the command line, and on platforms other than Windows.

To use the command, change to the directory that contains the amtldup command, then enter the command with the required options. For the location of the amtldup executable file on the relevant platform, see "Directory structure" on page 445 (AIX), page 455 (HP-UX), page 463 (Solaris), or page 468 (Windows).

The syntax of the amtldup command is: amtldup [options] -f file

where the options are:

-h *host* The TCP address of the LDAP server host (default, localhost).

-p *port* The port to connect to the LDAP server (default 389).

-D *dn* The user Id in Distinguished Name form. This user must have permission to create entries in the Directory Information Tree.

-w *password* The password for the user Id.

-Z Use a secure LDAP connection (SSL).

-K *keyfile* The name of the file that contains the encryption keys for use with SSL. This option is required if **-Z** is specified.

-P *key_pw* The password of the key file. This option is required if **-Z** is specified.

-N *key_name* The private key name to use in the key file. This option is required if **-Z** is specified.

-m mechanism The Simple Authentication and Security Layer (SASL) authentication mechanism to use to bind to the directory. If

specified, the mechanism must be one of:CRAM-MD5 for CRAM MD5 authentication

· GSSAPI for Kerberos authentication

EXTERNAL for any other user-provided SASL mechanism

-V *version* The version of the LDAP server (2 or 3; default is 3).

-b base_dn The base Distinguished Name under which new entries are created. The default is the container cn=AMIInfo under the default

directory naming context.

-v Use verbose mode.

-? Display syntax and options.

and file is the name of the repository xml file.

To delete AMI entries under the *base_dn*, use: amtldup [options] -X

Directory information tree

Entries in a directory are organized in a treelike structure called the Directory Information Tree (DIT). Within the DIT, the Distinguished Name (DN) of each entry determines its position in the tree. To store AMI information in the directory, you must select a base DN for the data. If the base DN does not already exist, it is created, then all the AMI data is placed under the base. Similarly, when the AMI searches for services and policies, you must specify the base DN from which to start the search. Typically, the contents of each XML repository file are stored under a distinct base DN (that is, in a separate container).

By default, the base DN is a container with the name cn=AMIInfo under the directory suffix.

The directory administrator must manage authorization to use the directory objects by using the usual administration utilities. The AMI does not provide tools to manage access control lists for service and policy information. When the entries are created, they inherit the access control list from the parent object. For further information, see "Security" on page 513.

Directory search

When an AMI application specifies a service or policy name, the AMI searches either the local repository file, or an LDAP directory. This depends on how the environment variables that AMI uses are set.

To search an LDAP directory for repository information, the AMT_REPOSITORY environment variable must be set to a reference to the repository information in LDAP URL format. This format is specified in RFC 2255 (for details, refer to the IETF Request for Comments Web page at http://www.ietf.org/rfc.html). For example:

AMT REPOSITORY=1dap://ldap.hursley.ibm.com:389/cn=Rep1,cn=AMIInfo,dc=hursley, o=ibm,c=uk

The LDAP URL can contain the bind DN, that is, the user Id with which to bind (this is needed for simple and CRAM-MD5 authentication). For example:

AMT REPOSITORY=1dap://ldap.hursley.ibm.com:389/cn=Rep1,cn=AMIInfo,dc=hursley, o=ibm,c=uk??sub??bindname=cn=Manager,o=MQSeries,o=ibm,c=uk

Alternatively, you can specify the bind DN separately using the environment variable AMT_LDAP_BINDDN, for example:

AMT LDAP BINDDN=cn=Manager,o=MQSeries,o=ibm,c=uk

If you specify the bind DN using both the LDAP URL and the AMT_LDAP_BINDDN environment variable, the environment variable takes precedence.

For C++ and Java applications, the LDAP URL can be specified by using the **AmSessionFactory setRepository** method. See "setRepository" on page 199 (C++) or "setRepository" on page 401 (Java).

To set security parameters, use the following environment variables:

AMT_LDAP_AUTHENTICATION

Set to one of none, simple, CRAM-MD5, or Kerberos according to the required security mechanism. If there is no bind DN set, the default is none, which implies anonymous access. Otherwise, the default is simple authentication.

AMT_LDAP_PASSWORD

Set to the password for the bind DN, if a password is required for the security mechanism that AMT_LDAP_AUTHENTICATION sets.

AMT_SSL

If this environment variable is set, Secure Sockets Layer (SSL) is used. Set it to the key file name, password, and key name (filename, password, name), if required. For example,

AMT SSL=keyfile, secret, cn=Manager, o=MQSeries, o=ibm, c=uk

Security

AMI administrators must consider the security of the bind DN. This is the user Id with which the AMI binds to the directory to extend, and search for, service and policy information. Authorization can range between full anonymous access, and the allocation of individual identities and access authorizations to each user.

For any chosen security strategy, the appropriate identities and passwords must be defined in the directory, and set by using the AMI environment variables on the client computers. You can set authorization to access the AMI information for individual users, or for groups, by using the normal directory utilities. You can use the Access Control List and inheritance mechanism of the directory to simplify authorization.

For SSL authentication or encryption, the IBM Global Security Kit (GSKit) V4 or later must be installed on each client machine that uses the AMI or the AMI Administration Tool). This is available as a free download from the IBM Web site,

http://www.ibm.com/software/network/directory

For Kerberos, the Kerberos client, IBM Network Authentication Service, must be installed. Note that Kerberos is not supported on Sun Solaris and HP-UX.

LDAP support

Chapter 21. 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 529
- "When your AMI program fails" on page 532

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:
 - On Windows, select **Start->Settings->Control Panel**, select **System**, select the **Environment** tab, then add or set the environment variables.
 - On UNIX systems, edit your .profile file.

When deciding where you want the trace files written, ensure that the user has sufficient authority to write to, not just read from, the disk.

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. 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=ves
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 E:\users\winn\build\bin\amtsosnd.exe <<< AMT trace >>> started at Wed May 30 09:07:10 2001

```
@(!) <<< *** Code Level is 1.3.0 *** >>>
   !(00330) BuildDate May 29 2001
   !(00330) Trace Level is 9
(00330)@09:07:10.513
  -->xmq xxxInitialize
   --->ObtainSystemCp
  !(00330) Code page is 437
  <----ObtainSystemCp (rc = 0)
  <--xmq xxxInitialize (rc = 0)
  -->amSessCreateX
   ---->amCheckAllBlanks()
  <---amCheckAllBlanks() (rc = 0)
  ---->amCheckValidName()
   <---amCheckValidName() (rc = 1)
   !(00330) Session name is: AMT.SAMPLE.SESSION
   !(00330) Allocating Object lock
   !(00330) Allocated object lock 008A2FC0
  !(00330) amLOCK_OBJECT_INIT(008A2FC0)
   --->amIdxTableAddEntry
   ---->amIdxTableCreate
  !(00330) allocating 1076
   !(00330) amLOCK_GLOBAL() 0
   !(00330) amLOCK OBJECT INIT(009BFD68)
   !(00330) amUNLOCK GLOBAL() 0
  <----amIdxTableCreate (rc = AM ERR OK)
   ---->amIdxTableLock
  !(00330) amLOCK OBJECT(009BFD68) 0
   <----amIdxTableLock (rc = AM ERR OK)
   ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(009BFD68) 0
  <----amIdxTableUnlock (rc = AM ERR OK)
  <---amIdxTableAddEntry (rc = AM ERR OK)
   ---->amSesClearErrorCodes
   ---->amIdxTableGetEntry
   ---->amIdxTableLock
  !(00330) amLOCK OBJECT(009BFD68) 0
   <----amIdxTableLock (rc = AM ERR OK)</pre>
   ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(009BFD68) 0
  <----amIdxTableUnlock (rc = AM_ERR_OK)</pre>
   <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 008A2780
   ---->amIdxTableGetEntry
   ---->amIdxTableLock
  !(00330) amLOCK OBJECT(009BFD68) 0
   <----amIdxTableLock (rc = AM ERR OK)
   ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(009BFD68) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
   <----amIdxTableGetEntry (rc = AM_ERR_OK)</pre>
   ---->amErrTranslate
  <----amErrTranslate (rc = 0)
  ---->amIdxTableGetEntry
   ---->amIdxTableLock
  !(00330) amLOCK OBJECT(009BFD68) 0
   <----amIdxTableLock (rc = AM ERR OK)</pre>
   ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(009BFD68) 0
   <----amIdxTableUnlock (rc = AM ERR OK)</pre>
   <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
   <----amSesClearErrorCodes (rc = 0)
```

```
---->amHashTableCreate()
  ---->amHashTableInit()
(00330)@09:07:10.604
  <----amHashTableInit() (rc = AM_ERR_OK)
  <---amHashTableCreate() (rc = AM ERR OK)
  ---->amHashTableCreate()
  ---->amHashTableInit()
  <----amHashTableInit() (rc = AM_ERR_OK)
  <----amHashTableCreate() (rc = AM_ERR_OK)
  ---->amHashTableCreate()
  ---->amHashTableInit()
  <----amHashTableInit() (rc = AM ERR OK)
  <----amHashTableCreate() (rc = AM ERR OK)
  ---->amHashTableCreate()
  ---->amHashTableInit()
  <----amHashTableInit() (rc = AM ERR OK)
  <----amHashTableCreate() (rc = AM ERR OK)
  ---->amHashTableCreate()
  ---->amHashTableInit()
  <----amHashTableInit() (rc = AM ERR OK)
  <----amHashTableCreate() (rc = AM ERR OK)
  ---->amHashTableCreate()
  ---->amHashTableInit()
  <----amHashTableInit() (rc = AM ERR OK)</pre>
  <----amHashTableCreate() (rc = AM ERR OK)
  ---->amHashTableCreate()
  ---->amHashTableInit()
  <----amHashTableInit() (rc = AM ERR OK)
  <----amHashTableCreate() (rc = AM ERR OK)
  ---->amHashTableCreate()
  ---->amHashTableInit()
  <----amHashTableInit() (rc = AM ERR OK)
  <----amHashTableCreate() (rc = AM ERR OK)
  --->amMaSrvCreate
  !(00330) amLOCK GLOBAL() 0
  ---->amIdxTableCreate
  !(00330) allocating 1076
  !(00330) amLOCK GLOBAL() 1
  !(00330) amLOCK OBJECT INIT(0089B600)
  !(00330) amUNLOCK GLOBAL() 1
  <----amIdxTableCreate (rc = AM ERR OK)
  !(00330) amUNLOCK GLOBAL() 0
  !(00330) Service object created 0x899CB8
  ---->amIdxTableAddEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM_ERR_OK)</pre>
  <----amIdxTableAddEntry (rc = AM ERR OK)
  ---->amMaSrvClearErrorCodes
  ---->amIdxTableGetEntry
(00330)@09:07:10.754
  -----amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <-----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 00899CB8
  <----amMaSrvClearErrorCodes (rc = AM_ERR_OK)
  <---amMaSrvCreate (rc = AM ERR OK)
  ---->amMaSrvSetSessionHandle
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
```

```
!(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)
  ---->amIdxTableUnlock
  !(00330) amLOCK_OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 00899CB8
  <----amMaSrvSetSessionHandle (rc = AM ERR OK)
  ---->amHashTableAddHandle()
  ---->amHashTableAddElement()
  !(00330) Element [SYSTEM.DEFAULT.SENDER] slot [61]
  <----amHashTableAddElement() (rc = AM ERR OK)</pre>
  <----amHashTableAddHandle() (rc = AM ERR OK)
  --->amMaSrvCreate
  !(00330) Service object created 0x89BA48
  ---->amIdxTableAddEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM_ERR_OK)</pre>
  <----amIdxTableAddEntry (rc = AM ERR OK)</pre>
  ---->amMaSrvClearErrorCodes
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK_OBJECT(0089B600) 0
  <-----amIdxTableLock (rc = AM ERR OK)</pre>
   ---->amIdxTableUnlock
  !(00330) amLOCK_OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)
  <----amIdxTableGetEntry (rc = AM_ERR_OK)</pre>
  !(00330) Object pointer 0089BA48
  <----amMaSrvClearErrorCodes (rc = AM ERR OK)
  <----amMaSrvCreate (rc = AM ERR OK)
  ---->amMaSrvSetSessionHandle
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
(00330)@09:07:10.814
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 0089BA48
  <----amMaSrvSetSessionHandle (rc = AM ERR OK)
  ---->amHashTableAddHandle()
  ---->amHashTableAddElement()
  !(00330) Element [SYSTEM.DEFAULT.RESPONSE.SENDER] slot [69]
  <----amHashTableAddElement() (rc = AM ERR OK)</pre>
  <---amHashTableAddHandle() (rc = AM ERR OK)</pre>
  --->amMaSrvCreate
  !(00330) Service object created 0x89D390
  ---->amIdxTableAddEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableAddEntry (rc = AM ERR OK)</pre>
  ---->amMaSrvClearErrorCodes
  ---->amIdxTableGetEntrv
   ----->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
```

```
---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM_ERR_OK)
  !(00330) Object pointer 0089D390
  <----amMaSrvClearErrorCodes (rc = AM ERR OK)
  <---amMaSrvCreate (rc = AM ERR OK)
  ---->amMaSrvSetSessionHandle
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 0089D390
  <----amMaSrvSetSessionHandle (rc = AM ERR OK)
  ---->amHashTableAddHandle()
  ---->amHashTableAddElement()
  !(00330) Element [SYSTEM.DEFAULT.RECEIVER] slot [16]
  <----amHashTableAddElement() (rc = AM ERR OK)</pre>
  <----amHashTableAddHandle() (rc = AM_ERR_OK)
  --->amMaSrvCreate
  !(00330) Service object created 0x9C01B0
  ---->amIdxTableAddEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)
  <----amIdxTableAddEntry (rc = AM ERR OK)</pre>
(00330)009:07:10.824
  ---->amMaSrvClearErrorCodes
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 009C01B0
  <----amMaSrvClearErrorCodes (rc = AM ERR OK)
  <---amMaSrvCreate (rc = AM ERR OK)
  ---->amMaSrvSetSessionHandle
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 009C01B0
  <----amMaSrvSetSessionHandle (rc = AM ERR OK)
  ---->amHashTableAddHandle()
  ---->amHashTableAddElement()
  !(00330) Element [SYSTEM.DEFAULT.PUBLISHER] slot [34]
  <----amHashTableAddElement() (rc = AM ERR OK)</pre>
  <----amHashTableAddHandle() (rc = AM ERR OK)
  --->amMaSrvCreate
  !(00330) Service object created 0x9C1AF8
  ---->amIdxTableAddEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
```

```
<----amIdxTableLock (rc = AM ERR OK)
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM_ERR OK)</pre>
  <----amIdxTableAddEntry (rc = AM ERR OK)</pre>
  ---->amMaSrvClearErrorCodes
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK_OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM_ERR_OK)</pre>
  !(00330) Object pointer 009C1AF8
  <----amMaSrvClearErrorCodes (rc = AM ERR OK)</pre>
  <---amMaSrvCreate (rc = AM ERR OK)
  ---->amMaSrvSetSessionHandle
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM_ERR_OK)</pre>
(00330)@09:07:10.824
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM_ERR_OK)</pre>
  <----amIdxTableGetEntry (rc = AM_ERR_OK)</pre>
  !(00330) Object pointer 009C1AF8
  <----amMaSrvSetSessionHandle (rc = AM ERR OK)
  ---->amHashTableAddHandle()
  ---->amHashTableAddElement()
  !(00330) Element [SYSTEM.DEFAULT.SUBSCRIBER] slot [18]
  <----amHashTableAddElement() (rc = AM ERR OK)</pre>
  <---amHashTableAddHandle() (rc = AM ERR OK)
  --->amMaSrvCreate
  !(00330) Service object created 0x9C3440
  ---->amIdxTableAddEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableAddEntry (rc = AM ERR OK)
  ---->amMaSrvClearErrorCodes
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM_ERR_OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM_ERR_OK)</pre>
  !(00330) Object pointer 009C3440
  <----amMaSrvClearErrorCodes (rc = AM ERR OK)
  <---amMaSrvCreate (rc = AM ERR OK)
  ---->amMaSrvSetSessionHandle
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK_OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM_ERR_OK)
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 009C3440
  <----amMaSrvSetSessionHandle (rc = AM ERR OK)
```

```
---->amMaSrvSetSubReceiverHandle
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK_OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM_ERR_OK)</pre>
  !(00330) Object pointer 009C1AF8
  <---amMaSrvSetSubReceiverHandle (rc = AM ERR OK)
  --->amMaMsgCreate
  !(00330) amLOCK GLOBAL() 0
(00330)@09:07:11.1\overline{2}4
  ---->amIdxTableCreate
  !(00330) allocating 1076
  !(00330) amLOCK_GLOBAL() 1
  !(00330) amLOCK OBJECT INIT(009C6740)
  !(00330) amUNLO\overline{C}K GLOBAL() 1
  <----amIdxTableCreate (rc = AM ERR OK)
  !(00330) amUNLOCK GLOBAL() 0
  !(00330) message object created -[10243464]
  ---->amIdxTableAddEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(009C6740) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(009C6740) 0
  <----amIdxTableUnlock (rc = AM_ERR_OK)
  <----amIdxTableAddEntry (rc = AM_ERR_OK)</pre>
  ---->amMaMsgClearErrorCodes
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(009C6740) 0
  <-----amId\overline{x}TableLock (rc = AM ERR OK)
  ---->amIdxTableUnlock
  !(00330) amLOCK_OBJECT(009C6740) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM_ERR_OK)</pre>
  !(00330) Object pointer 009C4D88
  <----amMaMsgClearErrorCodes (rc = AM ERR OK)</pre>
  <---amMaMsgCreate (rc = AM ERR OK)
  ---->amMaMsgSetSessionHandle
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(009C6740) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(009C6740) 0
  <----amIdxTableUnlock (rc = AM ERR OK)
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 009C4D88
  <---amMaMsgSetSessionHandle (rc = AM ERR OK)
  ---->amHashTableAddHandle()
  ---->amHashTableAddElement()
  !(00330) Element [SYSTEM.DEFAULT.SEND.MESSAGE] slot [83]
  <----amHashTableAddElement() (rc = AM ERR OK)
  <---amHashTableAddHandle() (rc = AM_ERR_OK)
  --->amMaMsgCreate
  !(00330) message object created -[10259368]
  ---->amIdxTableAddEntry
  ---->amIdxTableLock
  !(00330) amLOCK_OBJECT(009C6740) 0
  <----amIdxTableLock (rc = AM_ERR_OK)
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(009C6740) 0
  <----amIdxTableUnlock (rc = AM ERR OK)
```

```
<----amIdxTableAddEntry (rc = AM ERR OK)</pre>
  ---->amMaMsgClearErrorCodes
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(009C6740) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(009C6740) 0
(00330)@09:07:11.3\overline{3}5
  <-----amIdxTableUnlock (rc = AM_ERR_OK)</pre>
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 009C8BA8
  <----amMaMsgClearErrorCodes (rc = AM ERR OK)</pre>
  <---amMaMsgCreate (rc = AM_ERR_OK)
  ---->amMaMsgSetSessionHandle
  ---->amIdxTableGetEntry
  \verb|-----> amIdxTableLock|
  !(00330) amLOCK OBJECT(009C6740) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(009C6740) 0
  <----amIdxTableUnlock (rc = AM_ERR_OK)</pre>
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 009C8BA8
  <----amMaMsgSetSessionHandle (rc = AM ERR OK)
  ---->amHashTableAddHandle()
  ---->amHashTableAddElement()
  !(00330) Element [SYSTEM.DEFAULT.RECEIVE.MESSAGE] slot [17]
  <----amHashTableAddElement() (rc = AM_ERR_OK)</pre>
  <----amHashTableAddHandle() (rc = AM ERR OK)
  --->amMaPolCreate
  !(00330) amLOCK GLOBAL() 0
  ---->amIdxTableCreate
  !(00330) allocating 1076
  !(00330) amLOCK GLOBAL() 1
  !(00330) amLOCK_OBJECT_INIT(0089F8F0)
  !(00330) amUNLOCK_GLOBAL() 1
  <----amIdxTableCreate (rc = AM ERR OK)
  !(00330) amUNLOCK GLOBAL() 0
  !(00330) policy object created.
  !(00330) policy object initialized.
  ---->amIdxTableAddEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089F8F0) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089F8F0) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableAddEntry (rc = AM ERR OK)</pre>
  ---->amMaPolClearErrorCodes
  ---->amIdxTableGetEntry
   ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089F8F0) 0
  <-----amIdxTableLock (rc = AM_ERR_OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089F8F0) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM_ERR_OK)</pre>
  !(00330) Object pointer 009CB968
  <----amMaPolClearErrorCodes (rc = AM ERR OK)
  <---amMaPolCreate (rc = AM ERR OK)
  ---->amMaPolSetSessionHandle
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089F8F0) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
```

```
!(00330) amLOCK OBJECT(0089F8F0) 0
  <----amIdxTableUnlock (rc = AM ERR OK)
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 009CB968
(00330)@09:07:11.435
  <---amMaPolSetSessionHandle (rc = AM ERR OK)
  ---->amHashTableAddHandle()
  ---->amHashTableAddElement()
  !(00330) Element [SYSTEM.DEFAULT.POLICY] slot [29]
  <----amHashTableAddElement() (rc = AM ERR OK)</pre>
  <----amHashTableAddHandle() (rc = AM ERR OK)
  --->amMaPolCreate
  !(00330) policy object created.
  !(00330) Setting syncpoint on in policy
  !(00330) policy object initialized.
  ---->amIdxTableAddEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089F8F0) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089F8F0) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableAddEntry (rc = AM ERR OK)</pre>
  ---->amMaPolClearErrorCodes
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK_OBJECT(0089F8F0) 0
  <-----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK_OBJECT(0089F8F0) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM_ERR_OK)</pre>
  !(00330) Object pointer 009CC140
  <----amMaPolClearErrorCodes (rc = AM ERR OK)</pre>
  <---amMaPolCreate (rc = AM ERR OK)
  ---->amMaPolSetSessionHandle
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
  !(00330) amLOCK OBJECT(0089F8F0) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089F8F0) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM ERR OK)</pre>
  !(00330) Object pointer 009CC140
  <---amMaPolSetSessionHandle (rc = AM ERR OK)
  ---->amHashTableAddHandle()
  ---->amHashTableAddElement()
  !(00330) Element [SYSTEM.DEFAULT.SYNCPOINT.POLICY] slot [80]
  <----amHashTableAddElement() (rc = AM_ERR_OK)</pre>
  <---amHashTableAddHandle() (rc = AM ERR OK)
  ---->amMaSrvSetStringProp
  ---->amIdxTableGetEntry
  ----->amIdxTableLock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableLock (rc = AM ERR OK)</pre>
  ---->amIdxTableUnlock
  !(00330) amLOCK OBJECT(0089B600) 0
  <----amIdxTableUnlock (rc = AM ERR OK)</pre>
  <----amIdxTableGetEntry (rc = AM ERR OK)
  !(00330) Object pointer 00899CB8
  !(00330) [AMSRV STR Q NAME] set to [SYSTEM.DEFAULT.SENDER]
  <----amMaSrvSetStringProp (rc = AM_ERR_OK)
(00330)@09:07:11.445
  ---->amMaSrvSetStringProp
  ---->amIdxTableGetEntry
  ---->amIdxTableLock
```

```
!(00330) amLOCK OBJECT(0089B600) 0
<----amIdxTableLock (rc = AM ERR OK)
---->amIdxTableUnlock
!(00330) amLOCK_OBJECT(0089B600) 0
<----amIdxTableUnlock (rc = AM ERR OK)
<----amIdxTableGetEntry (rc = AM ERR OK)</pre>
!(00330) Object pointer 0089D390
!(00330) [AMSRV STR Q NAME] set to [SYSTEM.DEFAULT.RECEIVER]
<----amMaSrvSetStringProp (rc = AM_ERR_OK)
---->amMaSrvSetStringProp
---->amIdxTableGetEntry
---->amIdxTableLock
!(00330) amLOCK OBJECT(0089B600) 0
<----amIdxTableLock (rc = AM_ERR_OK)
---->amIdxTableUnlock
!(00330) amLOCK OBJECT(0089B600) 0
<----amIdxTableUnlock (rc = AM_ERR_OK)</pre>
<----amIdxTableGetEntry (rc = AM ERR OK)</pre>
!(00330) Object pointer 009C01B0
!(00330) [AMSRV_STR_Q_NAME] set to [SYSTEM.DEFAULT.PUBLISHER]
<----amMaSrvSetStringProp (rc = AM ERR OK)
---->amMaSrvSetStringProp
---->amIdxTableGetEntry
---->amIdxTableLock
!(00330) amLOCK OBJECT(0089B600) 0
<----amIdxTableLock (rc = AM ERR OK)</pre>
---->amIdxTableUnlock
!(00330) amLOCK OBJECT(0089B600) 0
<----amIdxTableUnlock (rc = AM_ERR_OK)
<----amIdxTableGetEntry (rc = AM_ERR_OK)
!(00330) Object pointer 009C1AF8
!(00330) [AMSRV_STR_Q_NAME] set to [SYSTEM.DEFAULT.SUBSCRIBER]
<----amMaSrvSetStringProp (rc = AM ERR OK)
---->amActivateFiles
!(00330) No DATAPATH specified from API
---->amGetDataPath()
!(00330) Using environment[E:\MQSeries\amt\]
<----amGetDataPath() (rc = 1)
!(00330) No POL HANDLER PATH specified from API
---->amGetPolHandlerPath()
---->amGetInstalledPath()
<----amGetInstalledPath() (rc = 1)
!(00330) Using installPath[E:\MQSeries\amt\handlers\]
<----amGetPolHandlerPath() (rc = 1)
!(00330) Policy Handler Path E:\MQSeries\amt\handlers\ from Installed Path Used
!(00330) No repository FILE specified from API
---->amGetRepositoryName()
!(00330) Using default[amt.xml]
<----amGetRepositoryName() (rc = 1)
!(00330) Repository 0x8A3670
!(00330) Repository ACTIVE
!(00330) No local host FILE specified from API
---->amGetLocalHostFileName()
!(00330) Using default[amthost.xml]
<----amGetLocalHostFileName() (rc = 1)
!(00330) Local Host 0x8A3620
!(00330) Local Host File ACTIVE
<---amActivateFiles (rc = 1)
!(00330) Using repository.
--->amRpsCreate
!(00330) amLOCK GLOBAL() 0
!(00330) Creating Repository Cache
---->CRpsTree::CRpsTree
<-----CRpsTree::CRpsTree (rc = 0)</pre>
!(00330) amUNLOCK GLOBAL() 0
<---amRpsCreate (rc = AM ERR OK)
--->amRpsGetIsOpen
```

```
!(00330) Object handle[9056960]
(00330)@09:07:11.845
  <---amRpsGetIsOpen (rc = AM ERR OK)
  --->amRpsOpen
  !(00330) amLOCK GLOBAL() 0
  !(00330) Object handle[9056960]
  ---->CRpsTree::refresh
  ----->CRpsTree::clearContent
  <-----CRpsTree::clearContent (rc = 0)</pre>
  !(00330) repository file name: E:\MQSeries\amt\amt.xml
  ---->CRpsTree::resolveRef
  ------CRpsTree::rsvService
  <-----CRpsTree::rsvService (rc = 0)</pre>
  ---->CRpsTree::rsvService
  <-----CRpsTree::rsvService (rc = 0)</pre>
  ---->CRpsTree::rsvService
  <-----CRpsTree::rsvService (rc = 0)</pre>
  <-----CRpsTree::resolveRef (rc = 0)</pre>
  <----CRpsTree::refresh (rc = 0)</pre>
  !(00330) amUNLOCK_GLOBAL() 0
  \leftarrow AM ERR OK)
  ---->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:

```
Entry:
       hh:mm:ss.tttt ---->function name()
Exit:
       hh:mm:ss.tttt <----function name() (rc = n)</pre>
Data:
       ! information
```

Timestamps of entry and exit records are in local time, and are accurate to 1/10000 second. The function call depth is indicated for entry and exit records by the dashes in the '----' or '<----' prefixes; two dashes per call level. For exit records, 'n' indicates the reason code on completion of the function. The default is to trace up to a depth of two function call levels, but this can be varied for batch applications. See "Control of formatted trace".

This a sample fragment from a formatted trace:

```
13:26:58.3263 -->amSendMsg
13:26:58.3264 ---->amSesGetSenderHandle
  ! amHashTableGetHandle failed.
13:26:58.3266 < ---- amSesGetSenderHandle (rc = [18][0x12])
13:26:58.3268 ---->amSesGetDistListHandle
  ! amHashTableGetHandle failed.
13:26:58.3269 < ----amSesGetDistListHandle (rc = [18][0x12])
13:26:58.3270 ---->amSesCreateSender
```

For IMS, batch, or RRS-batch applications, formatted trace is directed to a dataset specified by the user. In the CICS environment, formatted trace entries are written to the current CICS trace destination as determined by the CICS administrator.

Control of formatted trace

For IMS, batch, or RRS-batch applications, formatted trace can be turned on by specifying a JCL 'DD' statement for DD name 'AMTTRACE'. This can be assigned to SYSOUT or to a DASD dataset. If assigned to SYSOUT, the trace records are written to a single spool file.

AMI formatted trace will not be started unless '//AMTTRACE DD' is specified.

If the trace dataset becomes full during an AMI session, the file will automatically be reopened and the trace will wrap.

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 and LDAP error codes" on page 537.

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 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 and LDAP error codes

This appendix contains a description of the AMRC_* reason codes, divided into three sections according to the value of the corresponding completion code. Within each section they are in alphabetic order. For a list of reason codes in numeric order, see "Appendix B. Constants and structures" on page 561.

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

This appendix also contains a description of the secondary error code values that can occur when you update an LDAP directory with AMI information from a repository file. These values might be displayed in an AMI Administration Tool error message, or returned from a GetLastError request for the AMI object concerned.

Reason code: OK

The following reason code is returned with completion code: AMCC_OK

AMRC NONE

The request was successful with no error or warning returned.

Reason code: Warning

The following reason codes are returned with completion code: AMCC WARNING

AMRC BACKED OUT

The unit of work has been backed out.

AMRC_BACKOUT_LIMIT_ERR

The backout count of a received message was found to have exceeded its backout limit. The message was returned to the application and was requeued to the backout-requeue queue.

AMRC_BACKOUT_REQUEUE_ERR

The backout count of a received message was found to have exceeded its backout limit. The message was returned to the application. It could not be requeued to the backout requeue queue.

AMRC_CCSID_NOT_SUPPORTED

OS/390 V2 R9 (or later) is required to enable AMI publish subscribe or message element support under CICS. Ensure that your Language Environment installation is set up to use Unicode character conversion. See "Unicode character conversion" on page 458 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.

Reason code (warning)

AMRC_ENCODING_MIXED

The message contains mixed values for integer, decimal and floating point encodings, one or more of which conflict. An encoding value of undefined was returned to the application.

AMRC_FILE_ALREADY_EXISTS

The AMI was unable to receive the file as the current file disposition is 'new', and a file with the same name already exists on your system. The first message of the file transfer is returned to the application. If this occurs, we recommend that the current unit of work is backed out. This will ensure that the messages received from the service are in a consistent state.

AMRC FILE FORMAT CONVERTED

The AMI received a file successfully, but needed to convert between different file types. An example is from an 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_POL_HANDLER_WARNING

A warning was returned from a policy handler library invocation that occurred while processing the application function call. The policy handler reason code can be obtained by the secondary reason code value returned from a getlastError request for the AMI object concerned.

AMRC_POLICY_NOT_IN_REPOS

The definition name that was specified when creating a policy was not found in the repository. The policy was created using default values.

AMRC PUBLISHER NOT IN REPOS

The definition name that was specified when creating a publisher was not found in the specified repository. The publisher was created using default values.

AMRC_RECEIVER_NOT_IN_REPOS

The definition name that was specified when creating a receiver was not found in the repository. The receiver was created using default values.

AMRC_REPOS_WARNING

A warning associated with the underlying repository data was reported.

AMRC_RFH2_FORMAT_ERR

The format of an MQRFH2 rules and formatting header of a received message was not valid.

AMRC_SENDER_NOT_IN_REPOS

The definition name that was specified when creating a sender was not found in the repository. The sender was created using default values.

AMRC SUBSCRIBER NOT IN REPOS

The definition name that was specified when creating a subscriber was not found in the repository. The subscriber was created using default values.

AMRC_TRANSPORT_WARNING

A warning was reported by the underlying (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_DTD_NOT_FOUND

An AMI dtd file (amt.dtd) was not found with the xml repository file in the same directory.

AMRC_DIST_LIST_NOT_UNIQUE

The specified name could not be resolved to a unique distribution list because more than one distribution list with that name exists.

AMRC_ELEM_COUNT_PTR_ERR

The specified element count pointer was not valid.

AMRC_ELEM_INDEX_ERR

The specified element index value was not valid.

AMRC_ELEM_NAME_LEN_ERR

The specified element name length value was not valid.

AMRC ELEM NAME PTR ERR

The specified element name pointer was not valid.

AMRC_ELEM_NOT_FOUND

The specified element was not found.

Reason code (failed)

AMRC_ELEM_PTR_ERR

The specified element pointer was not valid.

AMRC_ELEM_STRUC_ERR

The specified element structure was not valid. The structure id, version, or a reserved field contained 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.

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 requested operation could not be performed because the specified service (sender, receiver, publisher, or subscriber) was open.

AMRC_INVALID_MSG_NAME

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

Reason code (failed)

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.

AMRC_JAVA_CLASS_ERR

A class referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_CREATE_ERR

An unexpected error occurred when creating an AMI Java object. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_FIELD_ERR

A field referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_JNI_ERR

An unexpected error occurred when calling the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC JAVA METHOD ERR

A method referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_LDAP_ERR

An error was encountered accessing the AMI repository information in the LDAP directory, or communicating with the LDAP server. The LDAP error code can be obtained from the secondary reason code value that is returned from a GetLastError request for the AMI object concerned. See "LDAP error codes" on page 556.

AMRC_LIBRARY_DUP_FUNCTION

A policy handler library that is specified by the repository attempted to register a function with an invocation point value that it has already registered.

AMRC_LIBRARY_FUNCTION_PTR_ERR

A policy handler library that is specified by the repository attempted to register a function with an invalid function pointer value (for example, NULL).

AMRC_LIBRARY_INV_POINT_ERR

A policy handler library that is specified by the repository attempted to register a function with an invocation point value that was not valid.

AMRC_LIBRARY_NOT_FOUND

A policy handler library file name specified in the repository was not found in the handlers directory.

AMRC_JAVA_NULL_PARM_ERR

The AMI Java code detected a null parameter that is not valid. (Not applicable to the C and C++ programming languages).

AMRC MSG HANDLE ERR

The specified message handle was not valid.

AMRC MSG ID BUFF LEN ERR

The specified message id buffer length value was not valid.

AMRC MSG ID BUFF PTR ERR

The specified message id buffer pointer was not valid.

AMRC_MSG_ID_LEN_ERR

The specified message id length value was not valid.

AMRC_MSG_ID_LEN_PTR_ERR

The specified message id length pointer was not valid.

AMRC_MSG_ID_PTR_ERR

The specified message id pointer was not valid.

AMRC_MSG_NOT_FOUND

The specified message was not found, so the request was not carried out.

AMRC_MSG_NOT_UNIQUE

The specified name could not be resolved to a unique message because more than one message object with that name exists.

AMRC_MSG_TYPE_NOT_REPORT

The message is not a report message.

AMRC_MSG_TYPE_PTR_ERR

The specified message type pointer was not valid.

AMRC NAME BUFF LEN ERR

The specified name buffer length value was not valid.

AMRC_NAME_BUFF_PTR_ERR

The specified name buffer pointer was not valid.

Reason code (failed)

AMRC_NAME_LEN_PTR_ERR

The specified name length pointer was not valid.

AMRC_NEGATIVE_RECEIVE_BUFF_LEN

The value of the buffer length parameter that is specified on a receive message request was negative.

AMRC_NO_MSG_AVAILABLE

No message was available for a receive request after the specified wait time.

AMRC_NO_RESP_SERVICE

The publish request was not successful because a response receiver service is required for registration and was not specified.

AMRC_NOT_AUTHORIZED

The user is not authorized by the underlying transport layer to perform the specified request.

AMRC POL HANDLER ERR

An error was returned from a policy handler library invocation that occurred while processing the application function call. The policy handler reason code can be obtained by the secondary reason code value returned from a getlastError request for the AMI object concerned.

AMRC POLICY NOT FOUND

The specified policy was not found, so the request was not carried out.

AMRC POLICY NOT UNIQUE

The specified name could not be resolved to a unique policy because more than one policy with that name exists.

AMRC PRIMARY HANDLE ERR

The primary handle (that is, the first parameter) passed on the API call was not valid. The most probable reason for failure is that the handle passed is a synonym handle, which is not valid as the primary handle on any call to the AMI.

AMRC_PUBLISHER_NOT_UNIQUE

The specified name could not be resolved to a unique publisher because more than one publisher object with that name exists.

AMRC_Q_NAME_BUFF_LEN_ERR

The specified queue name buffer length value was not valid.

AMRC O NAME BUFF PTR ERR

The specified queue name buffer pointer was not valid.

AMRC_Q_NAME_LEN_ERR

The specified queue name length value was not valid.

AMRC_Q_NAME_LEN_PTR_ERR

The specified queue name length pointer was not valid.

AMRC Q NAME PTR ERR

The specified queue name pointer was not valid.

AMRC READ OFFSET ERR

The current data offset used for reading bytes from a message is not valid.

AMRC_RECEIVE_BUFF_LEN_ERR

The buffer length specified for receiving data was not valid.

AMRC RECEIVE BUFF PTR ERR

The buffer pointer specified for receiving data was not valid.

AMRC_RECEIVE_DISABLED

The specified request could not be performed because the service in the underlying transport layer is not enabled for receive requests.

AMRC_RECEIVER_NOT_UNIQUE

The specified name could not be resolved to a unique receiver because more than one receiver object with that name exists.

AMRC_REPORT_CODE_ERR

The specified report (or feedback) code value was not valid.

AMRC_REPORT_CODE_PTR_ERR

The specified report code pointer was not valid.

AMRC_REPOS_CACHE_ERR

A module was loaded for use as a host file cache, but the module does not appear to be a valid host cache.

AMRC REPOS ERR

An error was returned when initializing or accessing the repository. This can occur for any of the following reasons:

- The repository XML file (for instance, amt.xml) contains data that is not valid.
- The DTD file (amt.dtd) was not found or contains data that is not valid.
- The files needed to initialize the repository (located in directories intlFiles and locales) could not be located.

Check that the DTD and XML files are valid and correctly located, and that the path settings for the local host and repository files are correct.

AMRC REPOS FILENAME ERR

The repository file name was not valid. The value of the appropriate environment variable should be corrected.

AMRC_REPOS_NOT_FOUND

The repository file was not found. The value of the appropriate environment variable should be corrected.

AMRC_RESERVED_NAME_IN_REPOS

The name specified for creating an object was found in the repository and is a reserved name that is not valid in a repository. The specified object was not created.

AMRC_RESP_RECEIVER_HANDLE_ERR

The response receiver service handle specified when sending a request message was not valid.

AMRC_RESP_SENDER_HANDLE_ERR

The response sender service handle specified when receiving a request message was not valid.

AMRC_RFH_ALREADY_EXISTS

A publish, subscribe, or unsubscribe command could not be added to the message because the message already contained an RFH header. The message requires a reset first, to remove existing data.

AMRC SEND DATA PTR ERR

The buffer pointer specified for sending data was not valid.

AMRC_SEND_DATA_LEN_ERR

The data length specified for sending data was not valid.

Reason code (failed)

AMRC_SEND_DISABLED

The specified request could not be performed because the service in the underlying transport layer is not enabled for send requests.

AMRC SENDER COUNT PTR ERR

The specified distribution list sender count pointer was not valid.

AMRC_SENDER_NOT_UNIQUE

The specified name could not be resolved to a unique sender because more than one sender object with that name exists.

AMRC_SENDER_USAGE_ERR

The specified sender service definition type was not valid for sending responses. To be valid for sending a response, a sender service must not have a repository definition, must have been specified as a response service when receiving a previous request message and must not have been used for any purpose other than sending responses.

AMRC SERVICE ALREADY CLOSED

The specified (sender, receiver, distribution list, publisher or subscriber) service was already closed.

AMRC_SERVICE_ALREADY_OPEN

The specified (sender, receiver, distribution list, publisher or subscriber) service was already open.

AMRC SERVICE FULL

The specified request could not be performed because the service in the underlying transport has reached its maximum message limit.

AMRC SERVICE HANDLE ERR

The service handle specified for a sender, receiver, distribution list, publisher, or subscriber was not valid.

AMRC_SERVICE_NOT_FOUND

The specified (sender, receiver, distribution list, publisher, or subscriber) service was not found, so the request was not carried out.

AMRC_SERVICE_NOT_OPEN

The request failed because the specified (sender, receiver, distribution list, publisher or subscriber) service was not open.

AMRC_SESSION_ALREADY_CLOSED

The session was already closed (or terminated).

AMRC_SESSION_ALREADY_OPEN

The session was already open (or initialized).

AMRC_SESSION_EXPIRED

Under the IMS environment, the current session has been marked as expired. See "Writing IMS applications using AMI" on page 437 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 code value returned from a 'GetLastError' request for the AMI object concerned. For more information, see "Common causes of problems" on page 533.

AMRC_TRANSPORT_LIBRARY_ERR

An error occurred loading the transport library.

AMRC_TRANSPORT_NOT_AVAILABLE

The underlying transport layer is not available.

AMRC_UNEXPECTED_ERR

An unexpected error occurred.

AMRC_WAIT_TIME_ERR

The specified wait-time value was not valid.

AMRC WAIT TIME PTR ERR

The specified wait time pointer was not valid.

AMRC_WAIT_TIME_READ_ONLY

An attempt was made to set the wait time in a policy object for which the wait-time was read-only.

Reason Code: Failed (Extended C AMI functions)

The following reason codes are returned with completion code: AMCC_FAILED

They are returned only by the extended C AMI functions.

AMRC ACCEPT DIRECT ERR

The specified accept direct requests value was not valid.

AMRC_ACCEPT_DIRECT_PTR_ERR

The specified accept direct requests pointer was not valid.

AMRC_ACCEPT_TRUNCATED_ERR

The specified accept truncated value was not valid.

AMRC ACCEPT TRUNCATED PTR ERR

The specified accept truncated pointer was not valid.

AMRC_ANON_ERR

The specified anonymous value was not valid.

AMRC_ANON_PTR_ERR

The specified anonymous pointer was not valid.

AMRC_APPL_GROUP_BUFF_LEN_ERR

The specified application group buffer length value was not valid.

AMRC_APPL_GROUP_BUFF_PTR_ERR

The specified application group buffer pointer was not valid.

AMRC_APPL_GROUP_LEN_ERR

The specified application group length value was not valid.

AMRC_APPL_GROUP_LEN_PTR_ERR

The specified application group length pointer was not valid.

AMRC APPL GROUP PTR ERR

The specified application group pointer was not valid.

Reason code (failed)

AMRC_BIND_ON_OPEN_ERR

The specified bind on open value was not valid.

AMRC_BIND_ON_OPEN_PTR_ERR

The specified bind on open pointer was not valid.

AMRC CHL NAME BUFF LEN ERR

The specified channel name buffer length value was not valid.

AMRC_CHL_NAME_BUFF_PTR_ERR

The specified channel name buffer pointer was not valid.

AMRC_CHL_NAME_LEN_ERR

The specified channel name length value was not valid.

AMRC_CHL_NAME_LEN_PTR_ERR

The specified channel name length pointer was not valid.

AMRC_CHL_NAME_PTR_ERR

The specified channel name pointer was not valid.

AMRC_CLOSE_DELETE_ERR

The specified close delete value was not valid.

AMRC CLOSE DELETE PTR ERR

The specified close delete pointer was not valid.

AMRC_CON_HANDLE_ERR

The specified connection handle was not valid.

AMRC_CON_INT_PROP_ID_ERR

The specified connection integer property identifier was not valid.

AMRC CON STR PROP ID ERR

The specified connection string property identifier was not valid.

AMRC_CONTEXT_ERR

The specified message context value was not valid.

AMRC_CONTEXT_PTR_ERR

The specified message context pointer was not valid.

AMRC_CONVERT_ERR

The specified convert message value was not valid.

AMRC_CONVERT_PTR_ERR

The specified convert message pointer was not valid.

AMRC_COUNT_ERR

The specified backout or retry count value was not valid.

AMRC_COUNT_PTR_ERR

The specified backout or retry count pointer was not valid.

AMRC_CUST_PARM_BUFF_LEN_ERR

The specified custom parameter buffer length value was not valid.

AMRC_CUST_PARM_BUFF_PTR_ERR

The specified custom parameter buffer pointer was not valid.

AMRC CUST PARM LEN ERR

The specified custom parameter length value was not valid.

AMRC_CUST_PARM_LEN_PTR_ERR

The specified custom parameter length pointer was not valid.

AMRC_CUST_PARM_PTR_ERR

The specified custom parameter pointer was not valid.

AMRC_DLY_PERSISTENCE_ERR

The specified delivery persistence value was not valid.

AMRC DLY PERSISTENCE PTR ERR

The specified delivery persistence pointer was not valid.

AMRC_DST_SUPPORT_ERR

The specified distribution list support value was not valid.

AMRC_DST_SUPPORT_PTR_ERR

The specified distribution list support pointer was not valid.

AMRC_EXPIRY_ERR

The specified message expiry value was not valid.

AMRC_EXPIRY_PTR_ERR

The specified message expiry pointer was not valid.

AMRC_FILE_DISP_ERR

The specified file disposition value was not valid.

AMRC FILE DISP PTR ERR

The specified file disposition pointer was not valid.

AMRC_FILE_RCD_LEN_ERR

The specified file record length value was not valid.

AMRC_FILE_RCD_LEN_PTR_ERR

The specified file record length pointer was not valid.

AMRC GROUP ID BUFF LEN ERR

The specified group id group buffer length value was not valid.

AMRC_GROUP_ID_BUFF_PTR_ERR

The specified group id buffer pointer was not valid.

AMRC_GROUP_ID_LEN_ERR

The specified group id length value was not valid.

AMRC_GROUP_ID_LEN_PTR_ERR

The specified group id length pointer was not valid.

AMRC_GROUP_ID_PTR_ERR

The specified group id pointer was not valid.

AMRC_HANDLE_POISON_MSG_ERR

The specified handle poison message value was not valid.

AMRC_HANDLE_POISON_MSG_PTR_ERR

The specified handle poison message pointer was not valid.

AMRC_HANDLE_PTR_ERR

The specified handle pointer was not valid.

AMRC_IMPL_OPEN_ERR

The specified implicit open value was not valid.

AMRC IMPL OPEN PTR ERR

The specified implicit open pointer was not valid.

AMRC_INFORM_IF_RET_ERR

The specified inform if retained value was not valid.

Reason code (failed)

AMRC_INFORM_IF_RET_PTR_ERR

The specified inform if retained pointer was not valid.

AMRC_INTERVAL_ERR

The specified retry interval value was not valid.

AMRC INTERVAL PTR ERR

The specified retry interval pointer was not valid.

AMRC_INVALID_IF_CON_OPEN

The requested operation could not be performed because the specified connection was open.

AMRC_LEAVE_OPEN_ERR

The specified leave open value was not valid.

AMRC_LEAVE_OPEN_PTR_ERR

The specified leave open pointer was not valid.

AMRC LOCAL ERR

The specified publish or subscribe locally value was not valid.

AMRC_LOCAL_PTR_ERR

The specified publish or subscribe locally pointer was not valid.

AMRC MCD PARM BUFF LEN ERR

The specified MCD parameter buffer length value was not valid.

AMRC MCD PARM BUFF PTR ERR

The specified MCD parameter buffer pointer was not valid.

AMRC MCD PARM LEN ERR

The specified MCD parameter length value was not valid.

AMRC MCD PARM LEN PTR ERR

The specified MCD parameter length pointer was not valid.

AMRC_MCD_PARM_PTR_ERR

The specified MCD parameter pointer was not valid.

AMRC_MGR_NAME_BUFF_LEN_ERR

The specified queue manager name buffer length value was not valid.

AMRC_MGR_NAME_BUFF_PTR_ERR

The specified queue manager name buffer pointer was not valid.

AMRC MGR NAME LEN ERR

The specified queue manager name length value was not valid.

AMRC_MGR_NAME_LEN_PTR_ERR

The specified queue manager name length pointer was not valid.

AMRC_MGR_NAME_PTR_ERR

The specified queue manager name pointer was not valid.

AMRC MSG INT PROP ID ERR

The specified message integer property identifier was not valid.

AMRC MSG LEN ERR

The specified message length value was not valid.

AMRC_MSG_LEN_PTR_ERR

The specified message length pointer was not valid.

AMRC MSG STR PROP ID ERR

The specified message string property identifier was not valid.

AMRC_MSG_TYPE_ERR

The specified message type value was not valid.

AMRC_NEW_CORREL_ID_ERR

The specified new correlation id value was not valid.

AMRC_NEW_CORREL_ID_PTR_ERR

The specified new correlation id pointer was not valid.

AMRC_NEW_PUBS_ONLY_ERR

The specified new publications only value was not valid.

AMRC_NEW_PUBS_ONLY_PTR_ERR

The specified new publications only pointer was not valid.

AMRC_PERSISTENCE_ERR

The specified persistence value was not valid.

AMRC_PERSISTENCE_PTR_ERR

The specified persistence pointer was not valid.

AMRC_POLICY_INT_PROP_ID_ERR

The specified policy integer property identifier was not valid.

AMRC POLICY STR PROP ID ERR

The specified policy string property identifier was not valid.

AMRC_PRIORITY_ERR

The specified priority value was not valid.

AMRC_PRIORITY_PTR_ERR

The specified priority pointer was not valid.

AMRC PUB ON REQ ERR

The specified publish on request value was not valid.

AMRC_PUB_ON_REQ_PTR_ERR

The specified publish on request pointer was not valid.

AMRC_PUB_OTHERS_ONLY_ERR

The specified publish to others only value was not valid.

AMRC_PUB_OTHERS_ONLY_PTR_ERR

The specified publish to others only pointer was not valid.

AMRC_READ_ONLY_ERR

The specified wait time read only value was not valid.

AMRC_READ_ONLY_PTR_ERR

The specified wait time read only pointer was not valid.

AMRC_REMOVE_ALL_ERR

The specified remove all subscriptions value was not valid.

AMRC_REMOVE_ALL_PTR_ERR

The specified remove all subscriptions pointer was not valid.

AMRC_REPORT_OPTION_ERR

The specified report option value was not valid.

AMRC REPORT OPTION PTR ERR

The specified report option pointer was not valid.

AMRC_RETAIN_ERR

The specified retain publications value was not valid.

Reason code (failed)

AMRC_RETAIN_PTR_ERR

The specified retain publications pointer was not valid.

AMRC_SEGMENT_ERR

The specified segment message value was not valid.

AMRC SEGMENT PTR ERR

The specified segment message pointer was not valid.

AMRC_SEQ_NO_ERR

The specified sequence number value was not valid.

AMRC_SEQ_NO_PTR_ERR

The specified sequence number pointer was not valid.

AMRC_SET_NAME_INVALID

The specified name cannot be changed.

AMRC SHARED ERR

The specified open shared value was not valid.

AMRC_SHARED_PTR_ERR

The specified open shared pointer was not valid.

AMRC SND TYPE ERR

The specified sender type value was not valid.

AMRC_SND_TYPE_PTR_ERR

The specified sender type pointer was not valid.

AMRC_SPLIT_LOGICAL_ERR

The specified split logical value was not valid.

AMRC SPLIT LOGICAL PTR ERR

The specified split logical pointer was not valid.

AMRC_SRV_INT_PROP_ID_ERR

The specified service integer property identifier was not valid.

AMRC SRV STR PROP ID ERR

The specified service string property identifier was not valid.

AMRC_SRV_TYPE_ERR

The specified service type value was not valid.

AMRC_SRV_TYPE_PTR_ERR

The specified service type pointer was not valid.

AMRC_SUBS_POINT_BUFF_LEN_ERR

The specified subscription point buffer length value was not valid.

AMRC_SUBS_POINT_BUFF_PTR_ERR

The specified subscription point buffer pointer was not valid.

AMRC_SUBS_POINT_LEN_ERR

The specified subscription point length value was not valid.

AMRC_SUBS_POINT_LEN_PTR_ERR

The specified subscription point length pointer was not valid.

AMRC SUBS POINT PTR ERR

The specified subscription point pointer was not valid.

AMRC_SUPPRESS_REG_ERR

The specified suppress registration value was not valid.

AMRC_SUPPRESS_REG_PTR_ERR

The specified suppress registration pointer was not valid.

AMRC_SYNCPOINT_ERR

The specified sync point value was not valid.

AMRC SYNCPOINT PTR ERR

The specified sync point pointer was not valid.

AMRC_TCP_ADDR_BUFF_LEN_ERR

The specified TCP/IP address buffer length value was not valid.

AMRC_TCP_ADDR_BUFF_PTR_ERR

The specified TCP/IP address buffer pointer was not valid.

AMRC_TCP_ADDR_LEN_ERR

The specified TCP/IP address length value was not valid.

AMRC_TCP_ADDR_LEN_PTR_ERR

The specified TCP/IP address length pointer was not valid.

AMRC_TCP_ADDR_PTR_ERR

The specified TCP/IP address pointer was not valid.

AMRC TRP TYPE ERR

The specified transport type value was not valid.

AMRC_TRP_TYPE_PTR_ERR

The specified transport type pointer was not valid.

AMRC_TRUSTED_ERR

The specified trusted value was not valid.

AMRC TRUSTED PTR ERR

The specified trusted pointer was not valid.

AMRC_USE_CORREL_ID_ERR

The specified use correlation id value was not valid.

AMRC_USE_CORREL_ID_PTR_ERR

The specified use correlation id pointer was not valid.

AMRC_WAIT_WHOLE_GROUP_ERR

The specified wait for whole group value was not valid.

AMRC_WAIT_WHOLE_GROUP_PTR_ERR

The specified wait for whole group pointer was not valid.

LDAP error codes

1 Unable to allocate memory

Cause: Not enough memory available to perform the update.

Action: Close any other applications that are running on the

workstation.

2 Unable to access or read SSL keyfile

Cause: The Secure Sockets Layer key file cannot be located.

Action: If you require SSL encryption, check the name of the file is

specified correctly.

3 Invalid SSL keyfile password

Cause: The password for the Secure Sockets Layer key file is

incorrect.

Action: Correct the keyfile password.

4 Invalid SSL key name

Cause: The private key name to use in the Secure Sockets Layer

key file is incorrect.

Action: Ensure that the key name specified is correct. The key

name is the name or label associated with the private key/certificate pair stored in the key database.

5 SSL certificate expired

Cause: When establishing an SSL connection with the directory,

either the server or client certificate has expired.

Action: Replace the expired certificate.

6 Unable to initialize SSL

Cause: Cannot initialize the Secure Sockets Layer

Action: Check the IBM Global Security Kit (GSKit) is installed and

configured, and that the SSL keyfile, password and name

are specified correctly.

7 No GSS security context

Cause: The Generic Security Service (GSSAPI) security context

cannot be initialized.

Action: If the directory server requires authentication using

Kerberos, ensure that you have authenticated to the security server. Otherwise, select a different authentication

type.

8 Unsupported character set

Cause: The character set specified to the amtldup program is not

supported, or is invalid.

Action: Use the AMI Administration Tool to update the LDAP

directory, or use the default character set for your system.

9 SSL not available

Cause: The Secure Sockets Layer library cannot be loaded.

Action: Check the IBM Global Security Kit (GSKit) is installed and

configured.

10 SSL handshake failure

Cause: The AMI cannot connect to the Secure Sockets Layer server.

Action: Ensure that the directory server is available, then retry the

operation.

11 Invalid LDAP URL

Cause: The LDAP URL, specifying the LDAP server and base to

use, is not valid.

Action: Check the setting of the AMT_REPOSITORY environment

variable. To retrieve AMI information from an LDAP directory, this environment variable must specify a valid LDAP URL, as described in the Application Messaging

Interface manual.

12 LDAP operations or protocol failure

Cause: An operational failure occurred while the AMI was

performing an LDAP operation.

Action: This error might be caused by a network failure. Retry the

operation when you are sure the server is available on the

network.

13 Character conversion failure

Cause: This error should not occur.

Action: If the error reoccurs, contact your IBM service

representative.

14 Invalid credentials

Cause: The AMI cannot authenticate to the directory server.

Action: Ensure that the Bind DN and corresponding password are

specified correctly.

15 Missing or invalid bind DN

Cause: The user Id to use as the Bind DN is not specified, or is

invalid.

Action: Ensure that a valid Bind DN is provided. Note that the full

DN (distinguished name) of the directory entry for the user

must be specified.

16 Directory server is not available

Cause: The AMI cannot connect to the LDAP directory.

Action: Ensure the directory server is running, then retry the

operation.

LDAP error codes

17 Unable to initialize connection to directory

Cause: The AMI cannot initialize a connection to the LDAP

directory.

Action: Check that the LDAP Server Name (hostname) and port

are specified correctly, then retry the operation.

18 Unable to open connection to directory

Cause: The AMI cannot open a connection to the LDAP directory.

Action: Check that the LDAP Server Name (hostname) and port

are specified correctly, then retry the operation.

19 Simple bind to directory failed

Cause: The AMI cannot perform a simple bind to the directory.

Action: Ensure that the Bind DN and corresponding password

have been specified correctly.

20 SASL bind to directory failed

Cause: A SASL (Simple Authentication and Security Layer) bind to

the directory has failed.

Action: Ensure that the directory server supports the authentication

type requested.

21 Error on disconnecting from directory

Cause: An error occurred when the AMI was disconnecting from

the directory.

Action: No action is required.

22 Missing or invalid base DN

Cause: The distinguished name under which the AMI information

is placed in the directory information tree is missing or

invalid.

Action: Correct the Base DN specified. Note that when first

creating AMI information in the directory, a container with the specified DN will be created if it is not already present. However, the parent DN for the container must already be

present.

23 Unable to access directory information

Cause: The AMI cannot access information that should be

available from the directory.

Action: Check that network connection to the directory server is

still available, and that the directory server is still running.

24 LDAP URL contains invalid base DN

Cause: The LDAP URL contains an invalid base DN.

Action: Correct the base DN specified in the LDAP URL. The base

DN must specify the DN of a directory object under which

AMI information can be found.

25 LDAP URL contains invalid scope

Cause: The LDAP URL contains an invalid scope.

Action: Correct or omit the scope specified in the LDAP URL. The

recommended scope for AMI searches is sub, that is, search of the whole subtree under the base DN. This is also the

AMI default.

26 Insufficient authority

Cause: The user or user Id under which the AMI is accessing the

directory (this is normally specified using the bind DN) does not have authority to access or update a required part

of the directory.

Action: Either specify a bind DN that has sufficient rights to access

to create items in the directory, or ensure that the user Id in use has sufficient access rights to the specified base DN and entries beneath it. For retrieving information, read access is required. For updating AMI information, create

and write access is also required.

99 Unexpected LDAP error

Cause: An unexpected error has occurred while accessing the

LDAP directory.

Action: If the error reoccurs, contact your IBM service

representative.

100 Unable to initialize XML parser

Cause: The Extensible Markup Language (XML) parser cannot be

initialized.

Action: This error should not occur. If the error reoccurs, contact

your IBM service representative.

101 Error parsing repository file

Cause: When updating AMI information in the directory, an error

was detected in the XML repository file, or an AMI dtd file (amt.dtd) is not present in the same directory as the

repository xml file.

Action: Use the AMI Administration Tool to create, modify, and

update the directory, or ensure that amt.dtd is present in

the appropriate directory.

110 Unable to initialize common services

Cause: An internal AMI error has occurred.

Action: If the error reoccurs, contact your IBM service

representative.

111 No AMI information changed

Cause: An attempt to update AMI information in the directory

was made, but no entries were changed.

Action: No action is required.

LDAP error codes

Appendix B. Constants and structures

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

The constants and structures

The constants are grouped according to the parameter or field to which they relate. Names of the constants in a group begin with a common prefix of the form AMxxxx_, where xxxx represents a string of 0 through 4 characters that indicates the nature of the values defined in that group. Within each group, constants are listed in numeric (or alphabetic) order.

Character strings are shown delimited by double quotation marks; the quotation marks are not part of the value.

AMB (Boolean constants)

```
AMB_FALSE 0L
AMB_TRUE 1L
```

AMBRW (Browse constants)

```
AMBRW_UNLOCK
                                 1L
AMBRW LOCK
                                 2L
AMBRW FIRST
                                 4L
AMBRW NEXT
                                 8L
AMBRW CURRENT
                                16L
AMBRW RECEIVE CURRENT
                                32L
AMBRW DEFAULT
                             AMBRW NEXT
AMBRW LOCK NEXT
                     ( AMBRW LOCK + AMBRW_NEXT )
                     ( AMBRW LOCK + AMBRW FIRST )
AMBRW LOCK FIRST
AMBRW LOCK CURRENT
                     ( AMBRW LOCK + AMBRW CURRENT )
```

AMCC (Completion codes)

```
AMCC_OK OL
AMCC_WARNING 1L
AMCC_FAILED 2L
```

AMDEF (Service and policy definitions)

```
AMDEF_POL "AMT.SYSTEM.POLICY"

AMDEF_PUB "AMT.SYSTEM.PUBLISHER"

AMDEF_RCV "AMT.SYSTEM.RECEIVER"

AMDEF_RSP_SND "AMT.SYSTEM.RESPONSE.SENDER"

AMDEF_SND "AMT.SYSTEM.SENDER"

AMDEF_SUB "AMT.SYSTEM.SUBSCRIBER"

AMDEF_SYNC POINT_POL "AMT.SYSTEM.SYNCPOINT.POLICY"
```

AMDT (Definition type constants)

AMDT UNDEFINED	0L
AMDT_RESPONSE	1L
AMDT TEMP DYNAMIC	2L
AMDT DYNAMIC	3L
AMDT_PREDEFINED	4L

AMENC (Encoding constants)

```
AMENC NORMAL
                                 0L
AMENC REVERSED
                                 1L
AMENC_NORMAL_FLOAT 390
                                 2L
AMENC REVERSED FLOAT 390
                                 3L
AMENC_UNDEFINED
                                 4L
AMENC_NATIVE
                      AMENC NORMAL
                                    (UNIX)
AMENC NATIVE
                      AMENC REVERSED (WIN32)
AMENC_NATIVE
                      AMENC NORMAL FLOAT 390 (OS/390)
```

AMELEM (AMI C element structure)

This is used in C to add and retrieve AMI RFH name/value pair elements to and from a message.

Fields

• strucId (AMCHAR8)

Structure identifier. The value must be AMELEM_STRUC_ID. The constantAMELEM_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMELEM_VERSION_1.

groupBuffLen (AMLONG)

Reserved, must be zero.

groupLen (AMLONG)

Reserved, must be zero.

pGroup (AMSTR)

Reserved, must be NULL.

nameBuffLen (AMLONG)

Name buffer length specifies length of the buffer in which the name is to be returned.

• nameLen (AMLONG)

Name length. AMLEN_NULL_TERM specifies that the string is null-terminated.

pName (AMSTR)

Name.

valueBuffLen (AMLONG)

Value buffer length.

valueLen (AMLONG)

Value length. AMLEN_NULL_TERM specifies that the string is null-terminated.

• pPalue (AMSTR)

Value.

typeBuffLen (AMLONG)

Reserved, must be zero.

typeLen (AMLONG)

Reserved, must be zero.

pType (AMSTR)

Reserved, must be NULL.

AMFB (Feedback codes)

AMFB_NONE	0L
AMFB CODE EXPIRATION	258L
AMFB_CODE_COA	259L
AMFB CODE COD	260L

AMFMT (Format constants)

AMFMT NONE	п	П
AMFMT RF HEADER	"MQHRF	п
AMFMT STRING	"MQSTR	П
AMFMT RF2 HEADER	"MQHRF2	П

AMGF and AMGRP (Group status constants)

```
AMGF_IN_GROUP
                                        1L
AMGF FIRST
                                        2L
AMGF_LAST
                                        4L
AMGRP MSG NOT IN GROUP
                                       0L
AMGRP_FIRST_MSG_IN_GROUP ( AMGF_IN_GROUP | AMGF_FIRST )
AMGRP_MIDDLE_MSG_IN_GROUP AMGF_IN_GROUP
AMGRP_LAST_MSG_IN_GROUP ( AMGF_IN_GROUP | AMGF_LAST )
AMGRP_ONLY_MSG_IN_GROUP ( AMGF_IN_GROUP | AMGF_FIRST | AMGF_LAST )
```

AMH (Handle constants)

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

AMLEN (String length constants)

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

AMMCD (Message Content Descriptor tag names)

AMMCD MSG SERVICE DOMAIN	"mcd.Msd"
AMMCD_MSG_SET	"mcd.Set"
AMMCD MSG TYPE	"mcd.Type"
AMMCD MSG FORMAT	"mcd.Fmt"

AMMT (Message types)

AMMT_NONE	0L
AMMT REQUEST	1L
AMMT_REPLY	2L
AMMT REPORT	4L
AMMT_DATAGRAM	8L

AMPOINTER (Pointer definition)

AMPOINTER *

AMPS (Publish/subscribe)

Publish/Subscribe constants

Publish/subscribe tag names

AMPS_COMMAND	"MQPSCommand"
AMPS COMP CODE	"MQPSCompCode"
AMPS DELETE OPTIONS	"MQPSDelOpts"
AMPS ERROR ID	"MQPSErrorId"
AMPS_ERROR_POS	"MQPSErrorPos"

"MOPSParmId" AMPS PARAMETER ID AMPS PUBLICATION OPTIONS "MQPSPubOpts" AMPS TIMESTAMP "MQPSPubTime" AMPS_Q_MGR_NAME "MQPSQMgrName" AMPS_Q_NAME "MQPSQName" AMPS REASON "MQPSReason" "MQPSReasonText" AMPS REASON TEXT "MQPSRegOpts" AMPS REGISTRATION OPTIONS AMPS_SEQUENCE_NUMBER "MQPSSeqNum" "MQPSStreamName" AMPS_STREAM_NAME AMPS_STRING_DATA
AMPS_TOPIC "MQPSStringData" "MQPSTopic" AMPS USER ID "MQPSUserId" AMPS_FILTER "MQPSFilter" AMPS SUBSCRIPTION POINT "MQPSSubPoint" "MQPSSequence" AMPS SEQUENCE "MQPSControl" AMPS CONTROL

Publish/subscribe tag values

AMPS ANONYMOUS "Anon" AMPS CORREL ID AS ID "CorrelAsId" AMPS DEREGISTER_ALL "DeregAll" AMPS DIRECT REQUESTS "DirectReg" AMPS INCLUDE STREAM NAME "InclStreamName" "InformIfRet" AMPS_INFORM_IF_RETAINED AMPS_LOCAL "Local" AMPS_NEW_PUBS_ONLY "NewPubsOnly" AMPS_PUB_ON_REQUEST ONLY "PubOnRegOnly" "DeletePub" AMPS DELETE PUBLICATION AMPS_DEREGISTER_PUBLISHER "DeregPub" AMPS_DEREGISTER_SUBSCRIBER "DeregSub" AMPS PUBLISH "Publish" AMPS REGISTER PUBLISHER "RegPub" AMPS REGISTER SUBSCRIBER "RegSub" AMPS REQUEST UPDATE "ReqUpdate" AMPS_IS_RETAINED_PUBLICATION "IsRetainedPub" AMPS_NO_REGISTRATION "NoReg" AMPS NONE "None' AMPS OTHER_SUBSCRIBERS_ONLY "OtherSubsOnly" AMPS RETAIN PUBLICATION "RetainPub" AMPS PERSISTENT "Pers" AMPS NON PERSISTENT "NonPers" AMPS PERSISTENT AS PUBLISHER "PersAsPub" AMPS_PERSISTENT_AS_QUEUE "PersAsQueue" AMPS CC OK "0" "1" AMPS_CC_WARNING "2" AMPS CC ERROR

Other publish/subscribe constants

AMPS APPL TYPE	"OPT APP GRP "
AMPS MSG TYPE	"OPT MSG TYPE "

AMRC (Reason codes)

AMRC NONE	0
AMRC UNEXPECTED ERR	1
AMRC INVALID Q NAME	2
AMRC_INVALID_SENDER_NAME	3
AMRC_INVALID_RECEIVER_NAME	4
AMRC_INVALID_PUBLISHER_NAME	5
AMRC_INVALID_SUBSCRIBER_NAME	6
AMRC_INVALID_POLICY_NAME	7
AMRC_INVALID_MSG_NAME	8
AMRC_INVALID_SESSION_NAME	9

AMRC_INSUFFICIENT_MEMORY AMRC_WAIT_TIME_READ_ONLY AMRC_SERVICE_NOT_FOUND AMRC_MSG_NOT_FOUND	10 11 12 13 14 15 16 17 18 19
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	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	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 AMRC_SESSION_NOT_OPEN AMRC_DIST_LIST_INDEX_ERR AMRC_WAIT_TIME_ERR	50 51 52 53 54 55 56 57 58 59
AMRC_SERVICE_NOT_OPEN AMRC_HEADER_TRUNCATED AMRC_HEADER_INVALID AMRC_DATA_LEN_ERR AMRC_BACKOUT_REQUEUE_ERR AMRC_BACKOUT_LIMIT_ERR AMRC_COMMAND_ALREADY_EXISTS AMRC_UNEXPECTED_RECEIVE_ERR AMRC_UNEXPECTED_SEND_ERR	60 61 62 63 64 65 66 67 68

AMRC_SENDER_USAGE_ERR AMRC_MSG_TRUNCATED AMRC_CLOSE_SESSION_ERR AMRC_READ_OFFSET_ERR	
AMRC SENDER USAGE ERR	
	70
AMRC MSG TRUNCATED	71
ANDO OLOGO OFFICE OFFI	7 -
AMRC_CLOSE_SESSION_ERK	72
AMRC_READ_OFFSET_ERR	73
AMDC DELL ALDEADY EVICEC	74
AMRC_RFH_ALKEADY_EXISTS	/4
AMRC_READ_OFFSET_ERR AMRC_RFH_ALREADY_EXISTS AMRC_GROUP_STATUS_ERR	75
AMRC_MSG_ID_LEN_ERR	76
AMRC MSG ID PTR ERR	77
AMRC_MSG_ID_PTR_ERR AMRC_MSG_ID_BUFF_LEN_ERR	78
ANICONSOLID DOLL CENTRA	70
AMRC_MSG_ID_BUFF_PTR_ERR	79
AMRC_MSG_ID_LEN_PTR_ERR	80
AMDO CODDEL TO LEN EDD	
AMRC_CORREL_ID_LEN_ERR AMRC_CORREL_ID_PTR_ERR	81
AMRC CORREL ID PTR ERR	82
AMDC CODDEL ID DIEE LEN EDD	83
AMRC_CORREL_ID_BUFF_LEN_ERR	
AMRC CORREL ID BUFF PTR ERR	84
AMRC_CORREL_ID_BUFF_PTR_ERR AMRC_CORREL_ID_LEN_PTR_ERR AMRC_FORMAT_LEN_ERR	85
ATING_CONNEL_ID_LEN_I TK_ERR	
AMRC_FORMAT_LEN_ERR AMRC_FORMAT_PTR_ERR AMRC_FORMAT_BUFF_PTR_ERR	86
AMRC FORMAT PTR FRR	87
AMDC FORMAT DUFF DTD FDD	88
AMRC_FORMAI_BOFF_PIR_ERR	
AMRC_FORMAT_LEN_PTR_ERR	89
AMRC_FORMAT_BUFF_LEN_ERR	90
ANDO NAME DUES DED SER	
AMRC_NAME_BUFF_PIR_ERR	91
AMRC NAME LEN PTR ERR	92
AMDO NAME DUEF LEN EDD	
AMRC_NAME_BUFF_PTR_ERR AMRC_NAME_LEN_PTR_ERR AMRC_NAME_BUFF_LEN_ERR	93
AMRC O NAME LEN ERR	94
AMDC O NAME DTD EDD	95
ANIC_Q_NANL_FIR_LIK	
AMRC_Q_NAME_LEN_ERR AMRC_Q_NAME_LEN_ERR AMRC_Q_NAME_BUFF_PTR_ERR AMRC_Q_NAME_LEN_PTR_ERR AMRC_Q_NAME_BUFF_LEN_ERR AMRC_Q_NAME_BUFF_LEN_ERR	96
AMRC O NAME LEN PTR ERR	97
AMDC O NAME BUEF LEN EDD	98
AMRC_Q_NAME_DUFF_LEN_ERR	
AMRC_WAIT_TIME_PTR_ERR	99
AMRC CCSID PTR ERR	100
AMDC ENCODING DID EDD	101
APIRC_ENCODING_FIR_ERR	101
AMDC DEEN TYDE DTD EDD	
AMING DEFIN TIPE PIN ENN	102
AMDC CCSID FDD	102
AMRC_CCSID_PTR_ERR AMRC_ENCODING_PTR_ERR AMRC_DEFN_TYPE_PTR_ERR AMRC_CCSID_ERR	103
AMRC_CCSID_ERR AMRC_DATA_LEN_PTR_ERR	103 104
AMRC_CCSID_ERR AMRC_DATA_LEN_PTR_ERR AMRC_GROUP STATUS PTR_ERR	103 104
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR	103 104 105
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR	103 104 105 106
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR	103 104 105 106
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR	103 104 105 106
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR	103 104 105 106 107 108
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR	103 104 105 106 107 108
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR	103 104 105 106 107 108
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED	103 104 105 106 107 108 109
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED	103 104 105 106 107 108 109
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE	103 104 105 106 107 108 109
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT	103 104 105 106 107 108 109
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP	103 104 105 106 107 108 109
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP	103 104 105 106 107 108 109 110 111 112
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED	103 104 105 106 107 108 109 110 111 112 113
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL	103 104 105 106 107 108 109 111 112 113 114
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL	103 104 105 106 107 108 109 110 111 112 113
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED	103 104 105 106 107 108 109 110 111 112 113 114 115
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED	103 104 105 106 107 108 109 110 111 112 113 114 115 116
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR	103 104 105 106 107 108 109 111 112 113 114 115 116 117
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR	103 104 105 106 107 108 109 111 112 113 114 115 116 117
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR AMRC_SENDER_COUNT_PTR_ERR	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR	103 104 105 106 107 108 109 111 112 113 114 115 116 117
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR AMRC_SENDER_COUNT_PTR_ERR	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SENTCE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR AMRC_MULTIPLE_REASONS	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR AMRC_SENDER_COUNT_PTR_ERR AMRC_MULTIPLE_REASONS AMRC_NO_RESP_SERVICE	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR AMRC_SENDER_COUNT_PTR_ERR	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SENTCE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR AMRC_SENDER_COUNT_PTR_ERR AMRC_MULTIPLE_REASONS AMRC_NO_RESP_SERVICE AMRC_DATA_PTR_ERR	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_DATA_DISABLED AMRC_MULTIPLE_REASONS	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SERVICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_DATA_DISABLED AMRC_MULTIPLE_REASONS	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SENTICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR AMRC_SENDER_COUNT_PTR_ERR AMRC_MULTIPLE_REASONS AMRC_NO_RESP_SERVICE AMRC_DATA_PTR_ERR AMRC_DATA_BUFF_LEN_ERR AMRC_DATA_BUFF_LEN_ERR AMRC_DATA_BUFF_PTR_ERR AMRC_DATA_BUFF_PTR_ERR AMRC_DATA_BUFF_PTR_ERR	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SENTICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR AMRC_SENDER_COUNT_PTR_ERR AMRC_MULTIPLE_REASONS AMRC_NO_RESP_SERVICE AMRC_DATA_PTR_ERR AMRC_DATA_BUFF_LEN_ERR AMRC_DATA_BUFF_LEN_ERR AMRC_DATA_BUFF_PTR_ERR AMRC_DATA_BUFF_PTR_ERR AMRC_DATA_BUFF_PTR_ERR	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SENTICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_MULTIPLE_REASONS AMRC_MULTIPLE_REASONS AMRC_NO_RESP_SERVICE AMRC_DATA_DTR_ERR AMRC_DATA_BUFF_LEN_ERR AMRC_DATA_BUFF_LEN_ERR AMRC_DATA_BUFF_PTR_ERR AMRC_DEFN_TYPE_ERR AMRC_DEFN_TYPE_ERR AMRC_DEFN_TYPE_ERR	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
AMRC_DATA_LEN_PTR_ERR AMRC_GROUP_STATUS_PTR_ERR AMRC_DATA_OFFSET_PTR_ERR AMRC_RESP_SENDER_HANDLE_ERR AMRC_RESP_RECEIVER_HANDLE_ERR AMRC_NOT_AUTHORIZED AMRC_TRANSPORT_NOT_AVAILABLE AMRC_BACKED_OUT AMRC_INCOMPLETE_GROUP AMRC_SEND_DISABLED AMRC_SENTICE_FULL AMRC_NOT_CONVERTED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_RECEIVE_DISABLED AMRC_GROUP_BACKOUT_LIMIT_ERR AMRC_SENDER_COUNT_PTR_ERR AMRC_MULTIPLE_REASONS AMRC_NO_RESP_SERVICE AMRC_DATA_PTR_ERR AMRC_DATA_BUFF_LEN_ERR AMRC_DATA_BUFF_LEN_ERR AMRC_DATA_BUFF_PTR_ERR AMRC_DATA_BUFF_PTR_ERR AMRC_DATA_BUFF_PTR_ERR	103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124

AMRC_DATA_OFFSET_ERR	127
AMRC_DATA_OFFSET_ERR AMRC_FILE_SYSTEM_ERR AMRC_FILE_ALREADY_EXISTS	128 129
AMRC_REPORT_CODE_PTR_ERR AMRC_MSG_TYPE_PTR_ERR AMRC_FILE_FORMAT_CONVERTED	130
AMRC_MSG_TYPE_PTR_ERR	131
AMRC_FILE_FORMAT_CONVERTED	132
AMRC_FILE_TRUNCATED	133 134
AMRC_FILE_NOT_FOUND	134
AMRC_NOT_A_FILE AMRC_FILE_NAME_LEN_ERR	136
AMRC_FILE_NAME_PTR_ERR	137
AMRC REH2 FORMAT FRR	138
AMRC_RFH2_FORMAT_ERR AMRC_CCSID_NOT_SUPPORTED	139
AMRC_FILE_MSG_FORMAT_ERR AMRC_MSG_TYPE_NOT_REPORT AMRC_ELEM_STRUC_TYPE_ERR AMRC_ELEM_STRUC_TYPE_BUFF_ERR	140
AMRC_MSG_TYPE_NOT_REPORT	141
AMRC_ELEM_STRUC_TYPE_ERR	142
AMRC_ELEM_STRUC_TYPE_BUFF_ERR	143
AMRC_FILE_TRANSFER_INVALID	
AMRC_FILE_NOT_WRITTEN AMRC_FILE_FORMAT_NOT_SUPPORTED	145
AMRC_NEGATIVE_RECEIVE_BUFF_LEN	140
AMRC LIBRARY NOT FOUND	148
AMRC_LIBRARY_NOT_FOUND AMRC_LIBRARY_FUNCTION_PTR_ERR	149
AMRC_LIBRARY_INV_POINT_ERR	
AMRC_LIBRARY_DUP_FUNCTION AMRC_POLICY_HANDLER_ERR	151
AMRC_POLICY_HANDLER_ERR	152
AMRC_POLICY_HANDLER_WARNING	153
AMRC_REPORT_CODE_ERR	154
AMRC_INVALID_TRACE_LEVEL AMRC_CONN_NAME_NOT_FOUND	400
AMRC CONN NAME NOT FOUND	401
WADC HUCK EILE NUT EULND	402
AMRC HOST FILENAME ERR	403
	404
AMRC_POLICY_NOT_IN_REPOS AMRC_SENDER_NOT_IN_REPOS AMRC_RECEIVER_NOT_IN_REPOS	405
AMRC_SENDER_NOT_IN_REPOS	406 407
AMRC_RECEIVER_NOT_IN_REPOS	407
	400
AMRC_SUBSCRIBER_NOT_IN_REPOS	410
AMRC_RESERVED_NAME_IN_REPOS	411
AMRC_REPOS_FILENAME_ERR	414
AMRC REPOS WARNING	415
AMRC_REPOS_ERR	416
AMRC_REPOS_NOT_FOUND	418
AMRC_TRANSPORT_LIBRARY_ERR	419
AMRC_HOST_CACHE_ERR	420
AMRC_REPOS_CACHE_ERR AMRC_PRIMARY_HANDLE_ERR	421 422
AMRC_PRIMARY_HANDLE_ERR AMRC SESSION EXPIRED	422
AMRC_DTD_NOT_FOUND	424
AMRC LDAP ERR	425

The following AMRC values are returned exclusively by extended C AMI functions.

```
AMRC_ACCEPT_DIRECT_ERR 201
AMRC_ACCEPT_DIRECT_PTR_ERR 202
AMRC_ACCEPT_TRUNCATED_ERR 203
AMRC_ACCEPT_TRUNCATED_PTR_ERR 204
AMRC_ANON_ERR 205
```

tui 00	
AMRC ANON PTR ERR	206
AMRC_APPL_GROUP_BUFF_LEN_ERR	
AMDC ADDI CDOLLD BLIFF DTD EDD	208
AMDC ADDI CDOUD LEN EDD	200
AMRC_APPL_GROUP_BUFF_PTR_ERR AMRC_APPL_GROUP_LEN_ERR	209
AMDO ADDI ODGID LEN DED EDD	010
AMRC_APPL_GROUP_LEN_PTR_ERR	210
AMRC_APPL_GROUP_PIK_ERR	211
AMRC_APPL_GROUP_PTR_ERR AMRC_BIND_ON_OPEN_ERR AMRC_BIND_ON_OPEN_PTR_ERR	212
AMRC_BIND_ON_OPEN_PIR_ERR	213
AMRC_CHL_NAME_BUFF_LEN_ERR AMRC_CHL_NAME_BUFF_PTR_ERR AMRC_CHL_NAME_LEN_ERR	214
AMRC_CHL_NAME_BUFF_PTR_ERR	215
AMRC_CHL_NAME_LEN_ERR	216
AMRC CHL NAME LEN PIR ERR	217
AMRC_CHL_NAME_LEN_ERR AMRC_CHL_NAME_LEN_PTR_ERR AMRC_CHL_NAME_PTR_ERR	218
AMRC_CHL_NAME_PTR_ERR AMRC_CLOSE_DELETE_ERR	219
AMRC_CLOSE_DELETE_PTR_ERR	220
AMRC_CONTEXT_ERR	221
AMRC_CONTEXT_ERR AMRC_CONTEXT_PTR_ERR	222
APIRC CONVERT ERR	223
AMRC CONVERT PTR ERR	224
AMRC_CONVERT_PTR_ERR AMRC_COUNT_ERR	225
AMRC COUNT PTR FRR	226
	227
AMRC_CUST_PARM_BUFF_LEN_ERR AMRC_CUST_PARM_BUFF_PTR_ERR	228
AMRC_CUST_PARM_LEN_ERR	229
7o_0001_171_EEN_EIN	
AMDC CUST DADM LEN DTD EDD	230
AMDC CUST DADM DTD EDD	231
AMRC_CUST_PARM_LEN_PTR_ERR AMRC_CUST_PARM_PTR_ERR AMRC_DLY_PERSISTENCE_ERR	533
AMDO DIV DEDCICTENCE DED EDD	233
AMRC_DLY_PERSISTENCE_PTR_ERR AMRC_DST_SUPPORT_ERR	234
AMRC_DST_SUPPORT_ERR AMRC_DST_SUPPORT_PTR_ERR AMRC_EXPIRY_ERR	
AMRC_DST_SUPPORT_PTR_ERR	235
AMRC_EXPIRY_ERR AMRC_EXPIRY_PTR_ERR AMRC_FILE_DISP_ERR AMRC_FILE_DISP_PTR_ERR	236
AMRC_EXPIRY_PIR_ERR	237
AMRC_FILE_DISP_ERR	238
AMRC_FILE_DISP_PIR_ERR	239
AMRC_FILE_RCD_LEN_ERR AMRC_FILE_RCD_LEN_PTR_ERR AMRC_GROUP_ID_BUFF_LEN_ERR AMRC_GROUP_ID_BUFF_PTR_ERR	040
AMRC_FILE_RCD_LEN_ERR	240
AMRC_FILE_RCD_LEN_PIR_ERR	241
AMRC_GROUP_ID_BUFF_LEN_ERR	242
AMRC_GROUP_ID_BUFF_PTR_ERR	243
AMRC GROUP ID LEN ERR	244
AMRC_GROUP_ID_LEN_PTR_ERR	245
AMRC GROUP ID PTR ERR	246
AMRC_HANDLE_POISON_MSG_ERR	247
AMRC_HANDLE_POISON_MSG_PTR_ERI	R 248
AMRC_HANDLE_PTR_ERR	249
AMRC_IMPL_OPEN_ERR	250
AMRC IMPL OPEN PTR ERR	251
AMRC_INFORM_IF_RET_ERR AMRC_INFORM_IF_RET_PTR_ERR	252
AMRC INFORM IF RET PTR ERR	253
AMRC_INTERVAL_ERR	254
AMRC_INTERVAL_PTR_ERR	255
AMRC_INTERVAL_PTR_ERR AMRC_LEAVE_OPEN_ERR	256
AMRC_LEAVE_OPEN_PTR_ERR	257
AMRC LOCAL FRR	258
AMRC_LOCAL_ERR AMRC_LOCAL_PTR_ERR	259
	_55
AMRC MCD PARM BUFF LEN ERR	260
AMRC_MCD_PARM_BUFF_PTR_ERR	261
AMRC MCD PARM LEN ERR	262
AMRC MCD PARM LEN PTR ERR	
	263
AMRC_MCD_PARM_PTR_ERR	263 264

AMRC_MGR_NAME_BUFF_LEN_ERR AMRC_MGR_NAME_BUFF_PTR_ERR AMRC_MGR_NAME_LEN_ERR AMRC_MGR_NAME_LEN_PTR_ERR AMRC_MGR_NAME_PTR_ERR	265 266 267 268 269
AMRC_MSG_LEN_ERR AMRC_MSG_LEN_PTR_ERR AMRC_MSG_TYPE_ERR AMRC_NEW_CORREL_ID_ERR AMRC_NEW_CORREL_ID_PTR_ERR AMRC_NEW_PUBS_ONLY_ERR AMRC_NEW_PUBS_ONLY_PTR_ERR AMRC_PERSISTENCE_ERR AMRC_PERSISTENCE_PTR_ERR AMRC_PRIORITY_ERR	270 271 272 273 274 275 276 277 278 279
AMRC_PRIORITY_PTR_ERR AMRC_PUB_ON_REQ_ERR AMRC_PUB_ON_REQ_PTR_ERR AMRC_PUB_OTHERS_ONLY_ERR AMRC_PUB_OTHERS_ONLY_PTR_ERR AMRC_READ_ONLY_ERR AMRC_READ_ONLY_PTR_ERR AMRC_REMOVE_ALL_ERR AMRC_REMOVE_ALL_PTR_ERR AMRC_REPORT_OPTION_ERR	280 281 282 283 284 285 286 287 288 289
AMRC_REPORT_OPTION_PTR_ERR AMRC_RETAIN_ERR AMRC_SEGMENT_ERR AMRC_SEGMENT_ERR AMRC_SEQ_NO_ERR AMRC_SEQ_NO_PTR_ERR AMRC_SET_NAME_INVALID AMRC_SHARED_ERR AMRC_SHARED_PTR_ERR	290 291 292 293 294 295 296 297 298 299
AMRC_SND_TYPE_ERR AMRC_SND_TYPE_PTR_ERR AMRC_SRV_TYPE_ERR AMRC_SRV_TYPE_PTR_ERR AMRC_SPLIT_LOGICAL_ERR AMRC_SPLIT_LOGICAL_PTR_ERR AMRC_SUBS_POINT_BUFF_LEN_ERR AMRC_SUBS_POINT_BUFF_FTR_ERR AMRC_SUBS_POINT_LEN_ERR AMRC_SUBS_POINT_LEN_PTR_ERR	300 301 302 303 304 305 306 307 308 309
AMRC_SUBS_POINT_PTR_ERR AMRC_SUPPRESS_REG_ERR AMRC_SUPPRESS_REG_PTR_ERR AMRC_SYNCPOINT_ERR AMRC_TCP_ADDR_BUFF_LEN_ERR AMRC_TCP_ADDR_BUFF_PTR_ERR AMRC_TCP_ADDR_LEN_ERR AMRC_TCP_ADDR_LEN_ERR AMRC_TCP_ADDR_LEN_PTR_ERR AMRC_TCP_ADDR_LEN_PTR_ERR AMRC_TCP_ADDR_PTR_ERR	310 311 312 313 314 315 316 317 318 319
AMRC_TRP_TYPE_ERR AMRC_TRP_TYPE_PTR_ERR AMRC_TRUSTED_ERR AMRC_TRUSTED_PTR_ERR	320 321 322 323

AMRC USE CORREL ID ERR	324
AMRC USE CORREL ID PTR ERR	325
AMRC WAIT WHOLE GROUP ERR	326
AMRC WAIT WHOLE GROUP PTR ERR	327
AMRC CON INT PROP ID ERR	328
AMRC CON STR PROP ID ERR	329
AMRC MSG INT PROP ID ERR	330
AMRC MSG STR PROP ID ERR	331
AMRC POLICY INT PROP ID ERR	332
AMRC POLICY STR PROP ID ERR	333
AMRC SRV INT PROP ID ERR	334
AMRC SRV STR PROP ID ERR	335
AMRC INVALID IF CON OPEN	336
AMRC_CON_HANDLE_ERR	337

The following AMRC values are applicable only to the Java programming language.

AMRC_JAVA_FIELD_ERR	500
AMRC_JAVA_METHOD_ERR	501
AMRC_JAVA_CLASS_ERR	502
AMRC_JAVA_JNI_ERR	503
AMRC_JAVA_CREATE_ERR	504
AMRC_JAVA_NULL_PARM_ERR	505

AMSD (System default names and handle synonyms)

System default names and handles

Default names

AMSD POL	"SYSTEM.DEFAULT.POLICY"
AMSD_PUB	"SYSTEM.DEFAULT.PUBLISHER"
AMSD PUB SND	"SYSTEM.DEFAULT.PUBLISHER"
AMSD_RCV	"SYSTEM.DEFAULT.RECEIVER"
AMSD_RCV_MSG	"SYSTEM.DEFAULT.RECEIVE.MESSAGE"
AMSD_RSP_SND	"SYSTEM.DEFAULT.RESPONSE.SENDER"
AMSD_SND	"SYSTEM.DEFAULT.SENDER"
AMSD_SND_MSG	"SYSTEM.DEFAULT.SEND.MESSAGE"
AMSD_SESSION_NAME	"SYSTEM.DEFAULT.SESSION"
AMSD_SUB	"SYSTEM.DEFAULT.SUBSCRIBER"
AMSD_SUB_SND	"SYSTEM.DEFAULT.SUBSCRIBER"
AMSD_SUB_RCV	"SYSTEM.DEFAULT.SUBSCRIBER.RECEIVER"
AMSD_SYNC_POINT_POL	"SYSTEM.DEFAULT.SYNCPOINT.POLICY"

Default handle synonyms

AMSD RSP SND HANDLE	(AMHSND)-5L
AMSD RCV HANDLE	(AMHRCV)-6L
AMSD_POL_HANDLE	(AMHPOL)-7L
AMSD SYNC POINT POL HANDLE	(AMHPOL)-8L
AMSD SND MSG HANDLE	(AMHMSG)-9L
AMSD_RCV_MSG_HANDLE	(AMHMSG)-10L

AMWT (Wait time constant)

```
AMWT_UNLIMITED
                                -1L
```

C constants used by extended C AMI functions

This section lists C constants that are defined in amtxc.h and that are used exclusively by extended C AMI functions.

AMCON (Connection object integer property value constants)

AMCON_NO	AMPROP_FALSE
AMCON YES	AMPROP TRUE
AMCON TRP TYPE MQ CLIENT	1L
AMCON_TRP_TYPE_MQ_SERVER	2L
AMCON TRP TYPE MQ AUTO	3L
AMCON TRP TYPE OTHER	4L

AMCON_INT (Connection object integer property identifiers)

```
AMCON INT TRP TYPE
                              101L
AMCON_INT_CCSID
                              102L
AMCON_INT_DST_SUPPORT
                              103L
AMCON_INT_TRUSTED
                              104L
AMCON_INT_MQHOBJ
                              105L
AMCON INT MOHCON
                             106L
AMCON_INT_USER_CONTEXT
AMCON_INT_PROP_LAST
                            107L
                             AMCON INT USER CONTEXT
```

AMCON_STR (Connection object string property identifiers)

AMCON STR NAME	1001L
AMCON STR MGR NAME	1002L
AMCON STR REAL MGR NAME	1003L
AMCON STR CLIENT CHL NAME	1004L
AMCON_STR_SERVER_TCP_ADDR	1005L
AMCON STR PROP LAST	AMCON STR SERVER TCP ADDR

AMEI (Expiry interval constant)

```
AMEI UNLIMITED
                           (-1L)
```

AMH (Handle constants)

```
AMH INVALID HCON
                            (-1L)
                           (-1L)
AMH INVALID HOBJ
```

AMH (Handle property limit constants for all object types)

AMH MIN	0x80000000
AMH MAX	0x7FFFFFF

AMINT (Positive integer property limit constants for all object types)

```
AMINT PROP MIN
AMINT PROP MAX
                           999999991
```

AMINV (Invocation points)

1L
2L
3L
4L
5L
6L
7L
8L
9L

AMINV_RECEIVER_CLOSE AMINV_RECEIVER_RECEIVE AMINV_DIST_LIST_OPEN AMINV_DIST_LIST_CLOSE AMINV_DIST_LIST_SEND AMINV_HANDLE_POISON_MSG	10L 11L 12L 13L 14L 15L
AMINV_TRANSPORT_FIRST AMINV_PRE_MQBACK AMINV_POST_MQBACK AMINV_PRE_MQBEGIN AMINV_PRE_MQCLOSE AMINV_PRE_MQCLOSE AMINV_PRE_MQCMIT AMINV_PRE_MQCMIT AMINV_PRE_MQCONN AMINV_PRE_MQCONN AMINV_POST_MQCONNX AMINV_POST_MQCONXX AMINV_POST_MQCONX AMINV_POST_MQCONX AMINV_POST_MQDISC AMINV_PRE_MQPUT AMINV_POST_MQPUT AMINV_POST_MQPUT AMINV_POST_MQPUT AMINV_POST_MQPUT	1000L 1001L 1002L 1003L 1004L 1005L 1006L 1007L 1010L 1011L 1013L 1014L 1015L 1016L 1017L 1018L 1019L 1019L 1020L 1020L 1021L
AMINV_PRE_MQSET AMINV_POST_MQSET	1025L 1026L

AMLONG (Signed integer property limit constants for all object types)

AMLONG_MIN 0x80000000 AMLONG MAX 0x7FFFFFFF

AMMSG (Assemble message options for use with amxMsgAssemble)

AMMSG_AMO_FORMAT_RFH 1L
AMMSG_AMO_FORMAT_RFH2 2L
AMMSG_AMO_EXT_HDR 4L

AMMSG (Message object integer property value constants)

AMMSG_NO AMMSG_YES	AMPROP_FALSE AMPROP_TRUE
AMMSG_CCSID_INHERIT AMMSG_CCSID_EMBEDDED AMMSG_CCSID_DFT	(-2L) (-1L) 0L
AMMSG_SEQ_NO_FIRST	1L
AMMSG_ORIGINAL_LEN_UNDEFINE	D (-1L)

AMMSG_RO_EXCPTN_NO AMMSG_RO_EXCPTN_YES AMMSG_RO_EXCPTN_DATA AMMSG_RO_EXCPTN_FULL	0L 1L 2L 3L
AMMSG_RO_EXPIRY_NO AMMSG_RO_EXPIRY_YES AMMSG_RO_EXPIRY_DATA AMMSG_RO_EXPIRY_FULL	
AMMSG_RO_COA_NO AMMSG_RO_COA_YES	0L 1L
AMMSG_RO_COA_FULL	

AMMSG_INT (Message object integer property identifiers)

```
AMMSG INT BACKOUT COUNT
AMMSG INT CCSID
                                102L
AMMSG_INT_MSG_DATA_LEN
                                103L
AMMSG_INT_MSG_LEN
                                104L
AMMSG_INT_ORIGINAL_LEN
                                105L
AMMSG_INT_DATA_OFFSET
AMMSG_INT_INT_ENCODING
                                106L
                                107L
AMMSG_INT_DEC_ENCODING
                                108L
AMMSG_INT_FLOAT_ENCODING
                                109L
AMMSG INT EXPIRY
                                110L
AMMSG INT FB
                                111L
AMMSG INT MSG TYPE
                                112L
AMMSG_INT_SEQ_NO
                               113L
AMMSG_INT_PERSISTENT
                                114L
AMMSG_INT_PRIORITY
AMMSG_INT_RO_EXCEPTION
AMMSG_INT_RO_EXPIRY
                                115L
                                116L
                                117L
AMMSG INT RO COA
                                118L
AMMSG INT RO COD
                                119L
AMMSG INT RO COPY MSG ID
                                120L
AMMSG INT RO DISCARD
                                121L
AMMSG_INT_RO_PAN
                                122L
AMMSG_INT_RO_NAN
                                123L
AMMSG_INT_RO_NEW_MSG_ID
AMMSG_INT_SEGMENTS
                                124L
                                125L
AMMSG_INT_IN_GROUP
                                126L
AMMSG INT APPL CCSID
                                127L
AMMSG INT USER CONTEXT
                                128L
AMMSG_INT_PROP_LAST
                               AMMSG_INT_USER_CONTEXT
```

AMMSG_STR (Message object string property identifiers)

AMMSG STR NAME	1001L
AMMSG_STR_FORMAT	1002L
AMMSG STR CORREL ID	1003L
AMMSG_STR_MSG_ID	1004L
AMMSG_STR_GROUP_ID	1005L
AMMSG_STR_PROP_LAST	AMMSG_STR_GROUP_ID

AMOP (AMI operation codes)

AMOP	CONNECTION	OPEN	1L
AMOP	CONNECTION	CLOSE	2L
AMOP.	BEGIN	_	3L

```
AMOP ROLLBACK
                               4L
AMOP_COMMIT
AMOP_SENDER_OPEN
AMOP_SENDER_CLOSE
                               5L
                               6L
                              7L
AMOP SENDER SEND
                              8L
AMOP RECEIVER OPEN
                              9L
AMOP RECEIVER CLOSE
                              10L
AMOP RECEIVER RECEIVE
                              11L
AMOP_DIST_LIST_OPEN
                               12L
AMOP_DIST_LIST_CLOSE
                               13L
AMOP DIST LIST SEND
                               14L
AMOP HANDLE POISON MSG
                               15L
```

AMPOL (Policy object integer property value constants)

AMPOL_CONTEXT_PASS_ID	AMPROP_FALS AMPROP_TRUE 1L 2L 3L 4L
AMPOL_PRIORITY_AS_TRP	(-1L)
AMPOL_PERSISTENCE_AS_TRP AMPOL_PERSISTENCE_NO AMPOL_PERSISTENCE_YES	(-1L) 0L 1L
AMPOL_REPORT_DATA_NO AMPOL_REPORT_DATA_YES AMPOL_REPORT_DATA_FULL	1L 2L 3L
AMPOL_BIND_ON_OPEN_AS_TRP AMPOL_BIND_ON_OPEN_NO AMPOL_BIND_ON_OPEN_YES	(-1L) 0L 1L
	0L 1L 2L
AMPOL_FILE_DISP_NEW AMPOL_FILE_DISP_OVERWRITE AMPOL_FILE_DISP_APPEND	0L 1L 2L
AMPOL_DLY_PERSISTENCE_AS_PUB AMPOL_DLY_PERSISTENCE_AS_TRP AMPOL_DLY_PERSISTENCE_NO AMPOL_DLY_PERSISTENCE_YES	(-1L) 0L

AMPOL_INT (Policy object integer property identifiers)

```
AMPOL INT SYNCPOINT
                                101L
AMPOL_INT_CONTEXT
                                102L
AMPOL_INT_SND_IMPL_OPEN
                                103L
AMPOL_INT_SND_LEAVE_OPEN
AMPOL_INT_PRIORITY
AMPOL_INT_PERSISTENCE
                                104L
                                105L
                                106L
AMPOL_INT_EXPIRY
                                107L
AMPOL INT RETRY COUNT
                               108L
AMPOL INT RETRY INTERVAL
                              109L
AMPOL_INT_REPORT_DATA
                              110L
                              111L
AMPOL_INT_RO_EXCEPTION
AMPOL_INT_RO_EXPIRY
                              112L
AMPOL_INT_RO_COA
                               113L
AMPOL INT RO COD
                               114L
```

```
AMPOL INT RO COPY MSG ID
                               115L
AMPOL_INT_RO_DISCARD AMPOL_INT_NEW_CORREL_ID
                               116L
                               117L
AMPOL_INT_SEGMENT
                              118L
AMPOL_INT_SPLIT_LOGICAL
                              119L
AMPOL INT FILE RCD LEN
                              120L
AMPOL INT BIND ON OPEN
                              121L
AMPOL INT RCV IMPL OPEN
                               122L
AMPOL_INT_RCV_LEAVE_OPEN
                               123L
AMPOL_INT_CLOSE_DELETE
                               124L
AMPOL_INT_WAIT_INTERVAL
AMPOL_INT_WAIT_INTERVAL_RO
                               125L
                              126L
AMPOL INT CONVERT
                               127L
AMPOL_INT_WAIT_WHOLE_GROUP
                               128L
AMPOL INT HANDLE POISON MSG 129L
AMPOL_INT_ACCEPT_TRUNCATED
                              130L
AMPOL_INT_SHARED
                               131L
AMPOL_INT_FILE_DISP
                               132L
AMPOL_INT_RETAIN_PUBS
                               133L
AMPOL_INT_PUB_OTHERS_ONLY
AMPOL_INT_SUPPRESS_REG
                               134L
                               135L
AMPOL INT PUB LOCAL
                               136L
AMPOL INT ACCEPT DIRECT
                              137L
AMPOL INT PUB ANON
                              138L
AMPOL INT PUB USE CORREL ID 139L
AMPOL_INT_SUB_LOCAL
                          140L
AMPOL_INT_NEW_PUBS_ONLY
                              141L
AMPOL_INT_PUB_ON_REQ
                               142L
AMPOL_INT_INFORM_IF_RET
AMPOL_INT_REMOVE_ALL_SUBS
AMPOL_INT_SUB_ANON
                               143L
                               144L
                               145L
AMPOL_INT_SUB_USE_CORREL_ID 146L
AMPOL_INT_DLY_PERSISTENCE
                              147L
AMPOL INT USER CONTEXT
                               148L
AMPOL INT PROP LAST
                               AMPOL INT USER CONTEXT
```

AMPOL_STR (Policy object string property identifiers)

AMPOL STR NAME	1001L
AMPOL STR APPL GROUP	1002L
AMPOL STR SUBS POINT	1003L
AMPOL STR PROP LAST	AMPOL STR SUBS POINT

AMPROP (Integer property true/false constants)

AMPROP FALSE AMPROP TRUE

AMSRV (Message object integer property value constants)

AMSRV_NO AMSRV_YES	AMPROP_FALSE AMPROP TRUE
AMSRV_TYPE_RCV	1L
AMSRV_TYPE_SND AMSRV_TYPE_DST_SND	2L 3L
AMSRV_TYPE_PUB_SND	4L
AMSRV_TYPE_SUB_SND AMSRV TYPE SUB RCV	5L 6L
AMSRV_SND_TYPE_NATIVE	1L
AMSRV_SND_TYPE_MQINT_V1 AMSRV_SND_TYPE_MQINT_V2	2L 3L
AMSRV_SND_TYPE_RFH1	4L
AMSRV_SND_TYPE_RFH2	5L
AMSRV_CCSID_UNDEFINED	0L

AMSRV_INT (Service object integer property identifiers)

```
AMSRV INT TYPE
                              101L
AMSRV INT SND TYPE
                              102L
AMSRV INT DEFN TYPE
                              103L
AMSRV INT ENCODING
                              104L
                              105L
AMSRV_INT_CCSID
AMSRV_INT_NEXT_SND_HANDLE
                              106L
AMSRV_INT_SUB_RCV_HANDLE
AMSRV_INT_MQHOBJ
                              107L
                              108L
AMSRV_INT_USER_CONTEXT
                              109L
AMSRV INT PROP LAST
                              AMSRV INT USER CONTEXT
```

AMSRV_STR (Service object string property identifiers)

```
AMSRV STR NAME
AMSRV_STR_Q_NAME
                                 1002L
AMSRV_STR_MGR_NAME
                                 1003L
AMSRV_STR_MODEL_Q_NAME
                                 1004L
AMSRV_STR_DFT_MSG_FORMAT
AMSRV_STR_DYNAMIC_PREFIX
                                 1005L
                              1006L
                              1007L
AMSRV STR DFT MCD DOMAIN
AMSRV STR DFT MCD SET
                               1008L
AMSRV STR DFT MCD TYPE
                               1009L
AMSRV STR DFT MCD FORMAT 1010L
AMSRV_SIK_DFI_ROD_. :
AMSRV_STR_CUST_PARMS
1011L
AMSRV_STR_CUST_PARMS
AMSRV_STR_CUST_PARMS
```

AMSTR (Maximum string length constant for all object types)

AMSTR MAX LEN

AMTC (Trace control constants)

AMTC TEXT	0L
AMTC_FUNCTION_ENTRY	1L
AMTC FUNCTION EXIT	2L
AMTC_DEFAULT	AMTC_TEXT

C constants and AMI parameter structures used by policy handlers

This section lists C constants that are defined in amtphlc.h and that are used exclusively by policy handlers.

AMPH (Policy handler continuation codes)

AMPH CONTINUE AMPH COMPLETE 1L

AMPH (Policy handler transport types)

AMPH TRANSPORT TYPE MQ "MQSeries" AMPH TRANSPORT LEN MQ AMPH_TRANSPORT_TYPE_DEFAULT AMPH_TRANSPORT_TYPE_MQ

AMPH (Policy handler maximum lengths)

AMPH MAX TRANSPORT LENGTH 256L

AMRO (Receive options)

AMRO BROWSE 11 AMRO_RECEIVE 2L AMRO RECEIVE FILE 3L

AMPHBGN (AMI C begin parameter structure)

This is used with AMIP_BEGIN.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHBGN STRUC ID The constant AMPHBGN_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHBGN VERSION 1.

hCon (AMHCON)

The handle of the AMI connection object.

hPolicy (AMHPOL)

The handle of the AMI policy object.

AMPHCLC (AMI C close connection parameter structure)

This is used with AMIP_CONNECTION_CLOSED.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHCLC_STRUC_ID. The constant AMPHCLC_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHBGN_VERSION_1.

hCon (AMHCON)

The handle of the AMI connection object.

hPolicy (AMHPOL)

The handle of the AMI policy object.

AMPHCLD (AMI C close distribution list parameter structure)

This is used with AMIP_DIST_LIST_CLOSE.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHCLD_STRUC_ID. The constant AMPHCLD_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHCLD VERSION 1.

hDstList (AMHDST)

The handle of the AMI distribution list object.

hPolicy (AMHPOL)

The handle of the AMI policy object.

AMPHCLS (AMI C close service parameter structure)

This is used with AMIP_SENDER_CLOSE or AMIP_RECEIVER_CLOSE.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHCLS_STRUC_ID. The constant AMPHCLS STRUC ID ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHCLS_VERSION_1.

hService (AMHSRV)

The handle of the AMI (sender or receiver) service object.

hPolicy (AMHPOL)

The handle of the AMI policy object.

AMPHCMT (AMI C commit parameter structure)

This is used with AMIP COMMIT.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHCMT_STRUC_ID. The constant AMPHCMT_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHCMT VERSION 1.

hCon (AMHCON)

The handle of the AMI connection object.

hPolicy (AMHPOL)

The handle of the AMI policy object.

AMPHHPM (AMI C handle poison message parameter structure)

This is used with AMIP HANDLE POISON MESSAGE.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHHPM_STRUC_ID. The constant AMPHHPM STRUC ID ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHHPM_VERSION_1.

hService (AMHSRV)

The handle of the AMI (receiver) service object.

• hPolicy (AMHPOL)

The handle of the AMI policy object.

• pData (PAMBYTE)

Pointer message data.

• dataLen (AMLONG)

The length of the message data in bytes.

AMPHOPC (AMI C open connection parameter structure)

This is used with AMIP CONNECTION OPEN.

Fields

• strucId (AMCHAR8)

Structure identifier. The value must be AMPHOPC_STRUC_ID. The constant AMPHOPC STRUC ID ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHOPC VERSION 1.

hCon (AMHCON)

The handle of the AMI connection object.

• hPolicy (AMHPOL)

The handle of the AMI policy object.

AMPHOPD (AMI C open distribution list parameter structure)

This is used with AMIP_DIST_LIST_OPEN.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHOPD_STRUC_ID. The constant AMPHOPD_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHOPD_VERSION_1.

hDstList (AMHDST)

The handle of the AMI distribution list service object.

 hPolicy (AMHPOL) The handle of the AMI policy object.

AMPHOPS (AMI C open service parameter structure)

This is used with AMIP_SENDER_OPEN and AMIP_RECEIVER_OPEN.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHOPS_STRUC_ID. The constant AMPHOPS_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHOPS_VERSION_1.

hService (AMHSRV)

The handle of the AMI (sender or receiver) service object.

• hPolicy (AMHPOL)

The handle of the AMI policy object.

This is used with

Fields

strucId (AMCHAR8)

Structure identifier.

version (AMLONG)

Structure version number.

hCon (AMHCON)

The handle of the AMI connection object.

hPolicy (AMHPOL)

The handle of the AMI policy object.

AMPHPARM (AMI parameter union)

This is a union of the following parameter structures:

• bgnParms (AMPHBGN)

The parameter structure for Begin.

clcParms (AMPHCLC)

The parameter structure for Close Connection.

cldParms (AMPHCLD)

The parameter structure for Close Distribution List.

clsParms (AMPHCLS)

The parameter structure for Close Service.

• cmtParms (AMPHCMT)

The parameter structure for Commit.

hpmParms (AMPHHPM)

The parameter structure for Handle Poison Message.

• opcParms (AMPHOPC)

The parameter structure for Open Connection.

opdParms (AMPHOPD)

The parameter structure for Open Distribution List.

opsParms (AMPHOPS)

The parameter structure for Open Service.

• rbkParms (AMPHRBK)

The parameter structure for Rollback.

rcsParms (AMPHRCS)

The parameter structure for Receiver from Service.

sndParms (AMPHSND)

The parameter structure for Send To Distribution List.

• snsParms (AMPHSNS)

The parameter structure for Send to Service.

AMPHRBK (AMI C rollback parameter structure)

This is used with AMIP ROLLBACK.

Fields

• strucId (AMCHAR8)

Structure identifier. The value must be AMPHRBK_STRUC_ID. The constant AMPHRBK_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHRBK_VERSION_1.

hCon (AMHCON)

The handle of the AMI connection object.

• hPolicy (AMHPOL)

The handle of the AMI policy object.

AMPHRCS (AMI C receive from service parameter structure)

This is used with AMIP_RECEIVER_RECEIVE.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHRCS_STRUC_ID. The constant AMPHRCS_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHRCS_VERSION_1.

hService (AMHSRV)

The handle of the AMI (receiver) service object.

• hPolicy (AMHPOL)

The handle of the AMI policy object.

rcvOpts (AMLONG)

Receive options (AMRO_BROWSE or AMRO_RECEIVE).

• brwOpts (AMHPOL)

Browse options.

• hSelMsg (AMHMSG)

The handle of the selection message.

buffLen (AMLONG)

The length of the message data buffer in bytes.

• pDataLen (PAMLONG)

Pointer to length of message data.

hMsg (AMHMSG)

The handle of the message object.

hRespService (AMHSRV)

Response (sender) service handle.

AMPHSND (AMI C send to distribution list parameter structure)

This is used with AMIP_DIST_LIST_SEND.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHSND_STRUC_ID. The constant AMPHSND_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHSND_VERSION_1.

• hDstList (AMHDST)

The handle of the AMI (sender or receiver) service object.

hResponse (AMHSRV)

The handle of the response receiver service object.

dataLen (AMLONG)

The length of the message data in bytes.

• pData (PAMBYTE)

Pointer to message data.

hMsg (AMHMSG)

The handle of the message object.

AMPHSNS (AMI C send to service parameter structure)

This is used with AMIP_SENDER_SEND.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHSNS_STRUC_ID. The constant AMPHSNS_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHSNS_VERSION_1.

hService (AMHSRV)

The handle of the AMI (sender) service object.

hPolicy (AMHPOL)

The handle of the AMI policy object.

hResponse (AMHSRV)

The handle of the response receiver service object.

• hReqMsg (AMHMSG)

The handle of the request message.

dataLen (AMLONG)

The length of the message data in bytes.

• pData (PAMBYTE)

Pointer to message data.

• hMsg (AMHMSG)

The handle of the message object.

C constants and MQI parameter structures used by policy handlers

AMPHMQBACK (AMI C MQBACK parameter structure)

This is used with AMIP_PRE_MQBACK and AMIP_POST_MQBACK.

Fields

• strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQBACK_STRUC_ID. The constant AMPHMQBACK_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQBACK_VERSION_1.

• pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

• pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

pReason (PMQLONG)

Pointer to the MQSeries reason code.

AMPHMQBEGIN (AMI C MQBEGIN parameter structure)

This is used with AMIP_PRE_MQBEGIN and AMIP_POST_MQBEGIN.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQBEGIN_STRUC_ID. The constant AMPHMQBEGIN_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

• version (AMLONG)

Structure version number. The value must be AMPHMQBACK_VERSION_1.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

• pBeginOptions (PMQVOID)

Pointer to the MQBEGIN options.

• pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

pReason (PMQLONG)

AMPHMQCLOSE (AMI C MQCLOSE parameter structure)

This is used with AMIP_PRE_MQCLOSE and AMIP_POST_MQCLOSE.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQCLOSE_STRUC_ID. The constant AMPHMOCLOSE STRUC ID ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQCLOSE_VERSION_1.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

• pHobj (PMQHOBJ)

Pointer to the MQSeries object handle.

pOptions (PMQLONG)

Pointer to the MQCLOSE options.

pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

pReason (PMQLONG)

Pointer to the MQSeries reason code.

AMPHMQCMIT (AMI C MQCMIT parameter structure)

This is used with AMIP_PRE_MQCMIT and AMIP_POST_MQCMIT.

Fields

• strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQCMIT_STRUC_ID. The constant AMPHMQCMIT_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. Structure version number. The value must be AMPHMQCMIT_VERSION_1.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

pReason (PMQLONG)

AMPHMQCONN (AMI C MQCONN parameter structure)

This is used with AMIP_PRE_MQCONN and AMIP_POST_MQCONN.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQCONN_STRUC_ID. The constant AMPHMOCONN STRUC ID ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQCONN_VERSION_1.

pQMgrName (PMQCHAR)

Pointer to the MQSeries queue manager name.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

• pReason (PMQLONG)

Pointer to the MQSeries reason code.

AMPHMQCONNX (AMI C MQCONNX parameter structure)

This is used with AMIP PRE MQCONNX and AMIP POST MQCONNX.

Fields

• strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQCONNX_STRUC_ID. The constant AMPHMQCONNX_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQCONNX_VERSION_1.

• pQMgrName (PMQCHAR)

Pointer to the MQSeries queue manager name.

pConnectOpts (PMQCNO)

Pointer to the MQCONNX options.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

• pReason (PMQLONG)

AMPHMQDISC (AMI C MQDISC parameter structure)

This is used with AMIP_PRE_MQDISC and AMIP_POST_MQDISC.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQDISC_STRUC_ID. The constant AMPHMQDISC_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQDISC_VERSION_1.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

pReason (PMQLONG)

Pointer to the MQSeries reason code.

AMPHMQGET (AMI C MQGET parameter structure)

This is used with AMIP_PRE_MQGET and AMIP_POST_MQGET.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQGET_STRUC_ID. The constant AMPHMQGET_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQGET_VERSION_1.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

pHobj (PMQHOBJ)

Pointer to the MQSeries object handle.

pMsgDesc (PMQV0ID)

Pointer to the MQSeries message descriptor.

pGetMsgOpts (PMQVOID)

Pointer to the MQSeries get message options.

pBufferLength (PMQLONG)

Pointer to buffer length.

• ppBuffer (PPMQVOID)

Pointer to buffer pointer.

• pDataLength (PMQLONG)

Pointer to data length.

pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

pReason (PMQLONG)

AMPHMQINQ (AMI C MQINQ parameter structure)

This is used with AMIP_PRE_MQINQ and AMIP_POST_MQINQ.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQINQ_STRUC_ID. The constant AMPHMOINO STRUC ID ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQINQ_VERSION_1.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

pHobj (PMQHOBJ)

Pointer to the MQSeries object handle.

pSelectorCount (PMQLONG)

Pointer to selector count.

• pSelectors (PMQLONG)

Array of selectors.

pIntAttrCount (PMQLONG)

Pointer to integer attribute count.

• pIntAttrs (PMQLONG)

Array of integer attributes.

• pCharAttrLength (PMQLONG)

Pointer to the length of the character attribute buffer.

pCharAttrs (PMQCHAR)

Pointer to the character attribute buffer.

pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

pReason (PMQLONG)

Pointer to the MQSeries reason code.

AMPHMQOPEN (AMI C MQOPEN parameter structure)

This is used with AMIP_PRE_MQOPEN and AMIP_POST_MQOPEN.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQOPEN_STRUC_ID. The constant AMPHMQOPEN STRUC ID ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQOPEN_VERSION_1.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

pObjDesc (PMQVOID)

Pointer to the MQSeries object descriptor.

pOptions (PMQLONG)

Pointer to the MQOPEN options.

Constants and structures

pHobj (PMQHOBJ)

Pointer to the MQSeries object handle.

pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

pReason (PMQLONG)

Pointer to the MQSeries reason code.

AMPHMQPUT (AMI C MQPUT parameter structure)

This is used with AMIP_PRE_MQPUT and AMIP_POST_MQPUT.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQPUT_STRUC_ID. The constant AMPHMQPUT_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQPUT_VERSION_1.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

• phObj (PMQHOBJ)

Pointer to the MQSeries object handle.

pMsgDesc (PMQVOID)

Pointer to the MQSeries message descriptor.

pPutMsgOpts (PMQVOID)

Pointer to the MQSeries put message options.

pBufferLength (PMQLONG)

Pointer to buffer length.

• ppBuffer (PPMQVOID

Pointer to buffer pointer.

pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

pReason (PMQLONG)

Pointer to the MQSeries reason code.

AMPHMQPUT1 (AMI C MQPUT1 parameter structure)

This is used with AMIP_PRE_MQPUT1 and AMIP_POST_MQPUT1.

Fields

strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQPUT1 STRUC ID. The constant AMPHMQPUT1_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQPUT1_VERSION_1.

pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

• pObjDesc (PMQVOID)

Pointer to the MQSeries object descriptor

• pMsgDesc (PMQVOID)

Pointer to the MQSeries message descriptor.

pPutMsgOpts (PMQVOID)

Pointer to the MQSeries put message options.

• pBufferLength (PMQLONG)

Pointer to buffer length.

• ppBuffer (PPMQVOID

Pointer to buffer pointer.

• pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

• pReason (PMQLONG)

Pointer to the MQSeries reason code.

AMPHMQSET (AMI C MQSET parameter structure)

This is used with AMIP_PRE_MQSET and AMIP_POST_MQSET.

Fields

• strucId (AMCHAR8)

Structure identifier. The value must be AMPHMQSET_STRUC_ID. The constant AMPHMQSET_STRUC_ID_ARRAY is defined with the same value, but as an array of characters instead of a string.

version (AMLONG)

Structure version number. The value must be AMPHMQSET_VERSION_1.

• pHconn (PMQHCONN)

Pointer to the MQSeries connection handle.

pHobj (PMQHOBJ)

Pointer to the MQSeries object handle.

pSelectorCount (PMQLONG)

Pointer to selector count.

• pSelectors (PMQLONG)

Array of selectors.

pIntAttrCount (PMQLONG)

Pointer to integer attribute count.

pIntAttrs (PMQLONG)

Array of integer attributes.

• pCharAttrLength (PMQLONG)

Pointer to the length of the character attribute buffer.

pCharAttrs (PMQCHAR)

Pointer to the character attribute buffer.

pCompCode (PMQLONG)

Pointer to the MQSeries completion code.

• pReason (PMQLONG)

Pointer to the MQSeries reason code.

Constants and structures

Appendix C. Extended C AMI functions

This appendix lists the extended C AMI functions that are provided to use in policy handler libraries. These functions are supported only in C. The required constants and function prototype definitions are provided in amtxc.h. If required, you can also use these functions in normal applications.

The extended C AMI functions include functions to set or inquire the integer and string properties for the connection, service, message, and policy objects. Valid integer property values and ranges are listed in the following sections. All strings have a maximum length of AMSTR_MAX_LEN bytes, except for AMI object names, which have a maximum length of AMLEN_MAX_NAME_LEN. Some lengths might be restricted further when connected to MQSeries.

Unless otherwise stated, all properties described in the following sections are specified through the AMI repository (where it includes a definition for the object concerned).

Connection object properties

Connection object properties can be integer properties or string properties.

Connection integer properties

AMCON INT TRP TYPE

Transport type. Specifies which MQSeries library the AMI dynamically loads and uses. The initial value of this property is established during object creation, before any policy handler invocation.

Valid values are:

- AMCON_TRP_TYPE_MQ_CLIENT Use the MQSeries client library.
- AMCON TRP TYPE MQ SERVER Use the MQSeries server library.
- AMCON_TRP_TYPE_AUTO (the default) Automatically determine which library to use, based on whether the MQSeries server library can be located.
- AMCON_TRP_TYPE_OTHER (other) If the transport type is set to this value during any AMOP_CONNECTION_OPEN invocation, certain MQSeries range and value limits are relaxed when other AMI object properties are set. This value can be set from a policy handler, but cannot be specified using the AMI Administration Tool.

AMCON INT CCSID

Coded character set identifier of the connection. This reflects the coded character set identifier of the transport connection. It provides the default CCSID value used for message element strings (including topic and filter elements) that an application program adds to, or retrieves from, a message.

This property value is established by using an MOSeries MOINO call during AMOP_CONNECTION_OPEN processing, after the queue manager connection is established. To change this value, a policy handler must set it subsequently, but before AMOP_CONNECTION_OPEN processing completes.

Valid values are:

• AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-99999999)

AMCON INT DST SUPPORT

Native distribution list support. This defines whether native MQSeries distribution list support is available in the underlying message transport, and whether it is used.

This property value is established by using an MQSeries MQINQ call during AMOP_CONNECTION_OPEN processing, after the queue manager connection is established. To ensure that a policy handler is invoked separately for each sender service in a distribution list, this MQINQ call must be intercepted, or a policy handler must subsequently set the value to AMCON_NO before AMOP_CONNECTION_OPEN processing completes.

For a policy handler that replaces the underlying transport with something other than MQSeries, this property must be set to AMCON_NO during AMINV_CONNECTION_OPEN processing.

Valid values are:

AMCON NO

All processing operations treat the distribution list as a list of separate sender services.

AMCON_YES

Distribution list operations attempt to exploit native MQSeries distribution list support.

AMCON_INT_TRUSTED

Transport connection is trusted. To change this value, a policy handler must set it during AMINV_CONNECTION_OPEN invocation point processing.

Valid values are:

- AMCON NO
- · AMCON YES

Use a fast-path connection for trusted applications.

AMCON INT MOHOBI

The object handle of the MQSeries queue manager. This property value is established during AMOP_CONNECTION_OPEN/MQCONN(X) processing.

Valid values are:

AMH_MIN to AMH_MAX

AMCON INT MOHCON

The connection handle of the MQSeries queue manager. This property value is established during an AMOP_CONNECTION_OPEN or MQINQ invocation point after MQCONN(X). A value of AMH_INVALID_HCON indicates that the connection is closed.

Valid values are:

AMH_MIN to AMH_MAX

AMCON_INT_USER_CONTEXT

The user context handle. The AMI does not use this property. It is provided so that other programs, such as a policy handler, can associate their own context information with a connection object.

Valid values are:

AMH_MIN to AMH_MAX

Connection string properties

AMCON_STR_NAME

The name of the connection object. This property cannot be changed.

AMCON_STR_MGR_NAME

The name used to connect to the queue manager. This is obtained from the repository and local host files. It might be blank if the connection is to the default queue manager.

AMCON_STR_REAL_MGR_NAME

The name of the real queue manager to which the AMI is connected. This might be different to the value of AMCON_MGR_NAME.

This value is established during an AMOP_CONNECTION_OPEN or MQINQ invocation after MQCONN(X). It is used for certain name/value pair values in the RFH message header during the construction of a publish/subscribe message.

AMCON_STR_CLIENT_CHL_NAME

The MQSeries client name of the server-connection channel. This value is used during AMOP_CONNECTION_OPEN/MQCONN(X) processing for MQSeries client connections only.

AMCON_STR_SERVER_TCP_ADDR

The MQSeries Client TCP address of the MQSeries server. This value is used during AMOP_CONNECTION_OPEN/MQCONN(X) processing for MQSeries client connections only.

Message object properties

Message object properties can be integer properties or string properties.

Message integer properties

AMMSG INT BACKOUT CNT

Backout count. This specifies the number of times a message that is included in a unit-of-work is backed out during a receive.

When sending a message, this property is ignored. When receiving a message, this should be set by the message transport before calling amxMsgUpdated. This is equivalent to the BackoutCount field in the MQSeries message descriptor.

Valid values are:

AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMMSG INT CCSID

Coded character set identifier (CCSID). This specifies the CCSID of the message and provides the function performed by amMsgGetCCSID or amMsgSetCCSID.

If the message redefines the CCSID in one or more of its message headers, this property specifies the CCSID of the first message header.

When receiving a message, this should be set by the message transport before calling amxMsgUpdated. If the message transport performs data conversion, it should update this field to reflect the final CCSID of the converted message. This is equivalent to the CodedCharSetId field in the MQSeries message descriptor.

Valid values are:

- AMMSG_CCSID_INHERIT (Inherit) The CCSID value is inherited from the transport connection.
- AMMSG_CCSID_EMBEDDED (Embedded) All CCSID values are specified in the message data.
- AMMSG CCSID DFT to AMINT PROP MAX (that is, 0-99999999)

AMMSG_INT_MSG_DATA_LEN

Message data length (excluding message headers). This specifies the number of data bytes in the message after any RFH message headers.

When sending a message, the AMI updates this value as the application writes bytes to the message during message construction. When receiving a message, the AMI sets this value during amxMsgUpdated processing, when it is called by the message transport.

Valid values are:

AMINT_PROP_MIN to AMINT_PROP_MAX (0-99999999).

AMMSG INT MSG LEN

Message length (including message headers). This specifies the total length of the message, including any RFH and other message headers.

When sending a message, the AMI sets this during amxMsgAssemble processing when it is called by the message transport. When receiving a message, this is set by the message transport before calling amxMsgUpdated.

Valid values are:

AMINT_PROP_MIN to AMINT_PROP_MAX (0-999999999).

AMMSG_INT_ORIGINAL_LEN

Original length (for use with report messages). This is used only for report messages and specifies the length of the original message to which the report applies (if the original message was a segment, this is the length of the segment rather than the length of the logical message).

When sending a report message, the application can set this value. When receiving a message, the message transport sets this value before calling amxMsgUpdated. This is equivalent to the OriginalLen field of the MQSeries message descriptor.

Valid values are:

- AMMSG_ORIGINAL_LEN_UNDEFINED
- AMINT_PROP_MIN to AMINT_PROP_MAX (0-99999999).

AMMSG INT DATA OFFSET

Data offset. This is the offset in bytes from the start of the message data (after any message headers) at which the next read or write bytes occurs.

When constructing a message for sending, the AMI updates this value as the application writes bytes to the message. When receiving a message, the message transport should set this to zero by using amMsgReset before calling amxMsgUpdated.

Valid values are:

• AMINT PROP MIN to AMINT PROP MAX (0-99999999).

AMMSG INT INT ENCODING

Integer encoding. This specifies the integer encoding of the message.

When sending a message, the application can set this value by using amMsgSetEncoding. When receiving a message, the message transport sets this value before amxMsgUpdated is called. If the message transport performs data conversion, it should update this field to reflect the integer encoding of the converted message. This is equivalent to the integer part of the Encoding field in the MQSeries message descriptor.

Valid values are:

- AMENC_NORMAL (normal)
- AMENC_REVERSED (reversed)
- AMENC_UNDEFINED (undefined)

AMMSG_INT_DEC_ENC

Decimal encoding. This specifies the decimal encoding of the message.

When sending a message, the application can set this value by using amMsgSetEncoding. When receiving a message, the message transport sets this value before amxMsgUpdated is called. If the message transport performs data conversion, it should update this field to reflect the decimal encoding of the converted message. This is equivalent to the decimal part of the Encoding field in the MQSeries message descriptor.

Valid values are:

- AMENC_NORMAL (normal)
- AMENC REVERSED (reversed)
- AMENC UNDEFINED (undefined)

AMMSG INT FLOAT ENCODING

Float encoding. This specifies the floating point encoding of the message.

When sending a message, the application can set this value by using amMsgSetEncoding. When receiving a message, the message transport sets this value before amxMsgUpdated is called. If the message transport performs data conversion, it should update this field to reflect the floating point encoding of the converted message. This is equivalent to the floating point part of the Encoding field in the MQSeries message descriptor.

Valid values are:

- AMENC_NORMAL (IEEE format)
- AMENC_REVERSED (IEEE format reversed)
- AMENC_NORMAL_FLOAT_390 (390 format)
- AMENC_UNDEFINED (undefined)

AMMSG INT FB

Feedback or report code. This sets the feedback code for a message and provides the function performed by amMsgGetReportCode or amMsgSetReportCode.

When receiving a message, the message transport should set this value before calling amxMsgUpdated. This is equivalent to the Feedback field in the MOSeries message descriptor.

Valid values are:

• AMINT PROP MIN to AMINT PROP MAX (that is, 0-99999999)

AMMSG INT EXPIRY

Expiry interval. This specifies the expiry interval in (tenths of a second) of the message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport sets this value before amxMsgUpdated is called. This is equivalent to the Expiry field in the MQSeries message descriptor.

Valid values are:

- AMEI_UNLIMITED
- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMMSG_INT_MSG_TYPE

Message type. This sets the message type of a message and provides the function performed by amMsgGetType or amMsgSetType.

When receiving a message, the message transport should set this value before calling amxMsgUpdated. This is equivalent to the MsgType field in the MQSeries message descriptor.

Valid values are:

- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-99999999). This includes the following predefined values:
 - AMMT NONE (None)
 - AMMT_REQUEST (Request)
 - AMMT_REPLY (Reply)
 - AMMT REPORT (Report)
 - AMMT DATAGRAM (Datagram)

AMMSG INT SEQ NO

Sequence number. This specifies the sequence number of a message within a message group.

When sending a message, the message transport generates this information automatically, and this property is ignored. When receiving a message, the message transport should set this value before calling amxMsgUpdated. This is equivalent to the MsgSeqNumber field in the MQSeries message descriptor.

Valid values are:

1 to AMINT_PROP_MAX (that is, 1-99999999)

AMMSG INT PERSISTENT

Persistence. This specifies the persistence of the message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the Persistence field in the MQSeries message descriptor.

Valid values are:

- AMMSG_NO (Not persistent)
- AMMSG_YES (Persistent)

AMMSG_INT_PRIORITY

Priority. This specifies the priority of the message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the Priority field in the MQSeries message descriptor.

Valid values are:

AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMMSG INT RO EXCEPTION

Report option: exception report messages. This specifies whether or not exception report messages are required for this message.

When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the Exception report options part of the Report field in the MQSeries message descriptor.

Valid values are:

- AMMSG_RO_EXCPTN_NO (Not required)
- AMMSG_RO_EXCPTN_YES (Required)
- AMMSG_RO_EXCPTN_DATA (Required with data)
- AMMSG_RO_EXCPTN_FULL (Required with full data)

AMMSG INT RO EXPIRY

Report option: expiry report messages. This specifies whether or not expiry report messages are required for this message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the Expiry report options part of the Report field in the MQSeries message descriptor.

Valid values are:

- AMMSG_RO_EXPIRY_NO (Not required)
- AMMSG_RO_EXPIRY_YES (Required)
- AMMSG_RO_EXPIRY_DATA (Required with data)
- AMMSG_RO_EXPIRY_FULL (Required with full data)

AMMSG_INT_RO_COA

Report option: confirm-on-arrival report messages. This report option specifies whether or not confirm-on-arrival report messages are required for this message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the COA report options part of the Report field in the MQSeries message descriptor.

Valid values are:

- AMMSG RO COA NO (Not required)
- AMMSG_RO_COA_YES (Required)
- AMMSG_RO_COA_DATA (Required with data)
- AMMSG_RO_COA_FULL (Required with full data)

AMMSG INT RO COD

Report option: confirm-on-delivery report messages. This specifies whether or not confirm-on-delivery report messages are required for this message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the COD report options part of the Report field in the MQSeries message descriptor

Valid values are:

- AMMSG_RO_COD_NO (Not required)
- AMMSG_RO_COD_YES (Required)
- AMMSG_RO_COD_DATA (Required with data)
- AMMSG_RO_COD_FULL (Required with full data)

AMMSG INT RO COPY MSG ID

Report option: copy MessageId (to CorrelId). This specifies the whether or not the MessageId (rather than the CorrelId) from messages that are sent using this policy is copied to the Correlld of report (or reply) messages that are sent in response.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the Copy MessageId or CorrelId report options part of the Report field in the MQSeries message descriptor.

Valid values are:

- AMMSG_NO
- AMMSG_YES

AMMSG_INT_RO_DISCARD

Report option: disposition. This specifies the whether or not to discard messages that are sent using this policy and that cannot be delivered, rather than put them on the dead letter queue.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the Dead Letter Queue or Discard report options part of the Report field in the MQSeries message descriptor.

Valid values are:

- AMMSG_NO
- AMMSG_YES

AMMSG_INT_RO_PAN

Report option: positive-action-notification report messages. This specifies whether or not PAN report messages are required for this message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the PAN report options part of the Report field in the MQSeries message descriptor.

Valid values are:

- AMMSG NO (Not required)
- AMMSG YES (Required)

AMMSG INT RO NAN

Report option: negative-action-notification report messages. This specifies whether or not NAN report messages are required for this message.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the NAN report options part of the Report field in the MQSeries message descriptor.

Valid values are:

- AMMSG_NO (Not required)
- AMMSG_YES (Required)

AMMSG_INT_RO_NEW_MSG_ID

Report option: New MessageId. When sending a message, this specifies whether to generate a new MessageId for a report (or reply) message sent in response, or whether to use the MessageId from the original message.

When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the New MessageId or Pass MessageId report options part of the Report field in the MQSeries message descriptor.

Valid values are:

- AMMSG NO
- AMMSG YES

AMMSG_INT_SEGMENTS

Segmentation allowed. This specifies whether or not the message transport can optionally split this message into segments during a send.

When sending a message, the policy defines this information, and this property is ignored. When receiving a message, the message transport should set this value before <code>amxMsgUpdated</code> is called. This is equivalent to the Segmentation Allowed option of the MsgFlags field in the MQSeries message descriptor.

Valid values are:

- AMMSG_NO
- AMMSG_YES

AMMSG_INT_IN_GROUP

The message is a member of a group. This specifies whether or not the message is a member of a message group, and provides the function performed by amMsgGetGroupStatus or amMsgSetGroupStatus.

When sending a message, the application can set this value by. When receiving a message, the message transport should set this value before **amxMsgUpdated** is called. This is equivalent to the Segmentation Allowed option of the MsgFlags field in the MQSeries message descriptor.

Valid values are:

- · AMGRP MSG NOT IN GRP
- AMGRP_FIRST_MSG_IN_GRP
- AMGRP_MIDDLE_MSG_IN_GRP
- AMGRP_LAST_MSG_IN_GRP
- AMGRP_ONLY_MSG_IN_GRP

AMMSG_INT_APPL_CCSID

Application coded character set identifier (CCSID). This specifies the CCSID that the application uses to add or retrieve message element data to or from the message, and provides the function performed by **amMsggetElementCCSID** or **amMsgSetElementCCSID**. The application can set this value. By default, the application uses the message transport CCSID obtained from the connection.

Valid values are:

AMMSG_CCSID_DFT_MIN to AMINT_PROP_MAX (that is, 0-99999999)

AMMSG_INT_USER_CONTEXT

User context handle. The AMI does not use this property. It is provided so that other programs, such as a policy handler, can associate their own context information with a policy object.

Valid values are:

• AMH_MIN to AMH_MAX

Message string properties

AMMSG STR NAME

Name of the message object. This provides the function performed by amMsgGetName. This property cannot be changed.

AMMSG STR FORMAT

Format of the message. This specifies the message format and provides the function performed by amMsgGetFormat or amMsgSetFormat.

When sending a message, the application can set this value. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the Format field in the MQSeries message descriptor.

AMMSG_STR_CORREL_ID

Correlation identifier. This specifies the CorrelId and provides the function performed by amMsgGetCorrelId or amMsgSetCorrelId.

When sending a message, the application can set this value or, using the appropriate policy option, the message transport can generate this value automatically. When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the Correlld field in the MQSeries message descriptor.

AMMSG_STR_MSG_ID

Message identifier. This specifies the MessageId and provides the function performed by amMsgGetMsg.

When sending a message, either the message transport generates this value automatically, or the AMI sets this value (using information from a request message). When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the MsgId field in the MQSeries message descriptor.

AMMSG_STR_GROUP_ID

Message group identifier. This specifies the message Group Id.

When sending a message, either the message transport generates this value automatically, or the AMI sets this value (using information from a request message). When receiving a message, the message transport should set this value before amxMsgUpdated is called. This is equivalent to the MsgId field in the MQSeries message descriptor.

Policy object properties

Policy object properties can be integer properties or string properties.

Policy integer properties

AMPOL INT SYNCPOINT

Sync point selected. This specifies whether or not messages sent and received using this policy are included in a unit-of-work.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL INT CONTEXT

Message context. This controls the level of message context information from a message receiver that uses this policy that is passed on in an output message that is sent using this policy.

Valid values are:

- AMPOL_CONTEXT_NONE (No context)
- AMPOL_CONTEXT_AS_TRP (As transport)
- AMPOL_CONTEXT_PASS_ID (Pass identity context)
- AMPOL_CONTEXT_PASS_ALL (Pass all context)

AMPOL_INT_SND_IMPL_OPEN

Implicitly open sender. This specifies whether a sender service should be implicitly opened to complete a send operation successfully using this policy.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL_INT_SND_LEAVE_OPEN

Leave sender open. This specifies whether an implicitly opened sender service remains open after a send operation using this policy has successfully completed.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL_INT_PRIORITY

Message priority. This specifies the priority of messages sent using this policy.

Valid values are:

AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-999999999)

AMPOL_INT_PERSISTENCE

Message persistence. This specifies the persistence of messages sent using this policy.

Valid values are:

- AMPOL_PERSISTENCE_AS_TRP (as transport)
- AMPOL_PERSISTENCE_NO (not persistent)
- AMPOL_PERSISTENCE_YES (persistent)

AMPOL INT EXPIRY

Expiry interval. This specifies the expiry interval (in tenths of a second) of messages sent using this policy.

Valid values are:

- AMEI_UNLIMITED
- AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-99999999)

AMPOL_INT_RETRY_COUNT

Retry count. This specifies the number of retries to use when messages that are sent using this policy encounter a temporary error condition.

Valid values are:

AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-99999999)

AMPOL_INT_RETRY_INTERVAL

Retry interval. This specifies the time interval (in milliseconds) between retries when messages that are sent using this policy encounter a temporary error condition.

Valid values are:

AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-99999999)

AMPOL INT REPORT DATA

Data required in report messages. This specifies the amount of data, if any, required in report messages that are sent in response to messages that are sent using this policy.

Valid values are:

- AMPOL_REPORT_DATA_NO (no data)
- AMPOL_REPORT_DATA_YES (data)
- AMPOL REPORT DATA FULL (full data)

AMPOL_INT_RO_EXCEPTION

Report option: exception report messages required. This specifies whether or not exception report messages are required for messages that are sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_RO_EXPIRY

Report option: expiry report messages required. This specifies whether or not expiry report messages are required for messages that are sent using this policy.

Valid values are:

- AMPOL NO
- AMPOL YES

AMPOL INT RO COA

Report option: confirm-on-arrival report messages required. This specifies whether or not confirm-on-arrival report messages are required for messages that are sent using this policy.

Valid values are:

- AMPOL NO
- AMPOL YES

AMPOL_INT_RO_COD

Report option: confirm-on-delivery report messages required. This report option specifies whether or not confirm-on-delivery report messages are required for messages that are sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_RO_COPY_MSG_ID

Report option: copy MessageId to CorrelId. This specifies the whether or not to copy the MessageId (rather than CorrelId) from messages that are sent using this policy to the Correlld of report or reply messages that are sent in response.

Valid values are:

- AMPOL NO
- AMPOL YES

AMPOL INT RO DISCARD

Report option: disposition. This specifies the whether or not messages that are sent using this policy are discarded (rather than put to the dead letter queue) if they cannot be delivered.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL INT NEW CORREL ID

Generate new Correlld. This report option specifies whether or not to generate a new Correlld automatically for messages that are sent using this policy (except when sending a report or response message where the MessageId or CorrelId from the request message is always used).

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_SEGMENT

Segment message. This specifies whether or not messages that are sent using this policy that can be segmented for transmission by the underlying message transport.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL INT SPLIT LOGICAL

Split files on logical boundaries. This specifies whether or not files that are sent using this policy are split into separate messages on logical, rather than physical, boundaries, as determined by the file record length.

On Windows, HP-UX, AIX, and Sun Solaris, if the File Record Length is zero, this is the end of a line. On OS/390, this is a record boundary.

If physical splitting is used, files are split into separate messages on boundaries that the AMI determines, and the messages include sufficient data to recreate them (as files) at the receiver.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL_INT_FILE_RCD_LEN

File record length. This specifies the boundary used with logical file splitting to split a file into separate messages.

On Windows, HP-UX, AIX, and Sun Solaris, a value of zero specifies that the end of a line is used. On OS/390, a value of zero specifies that the record boundary is used.

Valid values are:

AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-99999999)

AMPOL_INT_BIND_ON_OPEN

Bind on open. This specifies when a sender service binds to its underlying message queue in a cluster environment.

Valid values are:

- AMPOL_BIND_ON_OPEN_AS_TRP (As transport)
- AMPOL_BIND_ON_OPEN_NO (Do not bind on open)
- AMPOL_BIND_ON_OPEN_YES (Bind on open)

AMPOL_INT_RCV_IMPL_OPEN

Implicitly open receiver. This specifies whether or not to open a receiver service implicitly in order to complete a receive operation successfully using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_RCV_LEAVE_OPEN

Leave receiver open. This specifies whether or not to leave an implicitly opened receiver service open after a receive operation using this policy has completed successfully.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL_INT_RCV_CLOSE_DELETE

Delete temporary dynamic queue on close. This specifies whether or not to delete a temporary dynamic queue when its receiver service is closed using this policy.

Valid values are:

• AMPOL_CLOSE_DELETE_NO

Do not delete the temporary dynamic queue.

AMPOL CLOSE DELETE YES

Delete the temporary dynamic queue if it is empty.

• AMPOL_CLOSE_DELETE_PURGE

If the temporary dynamic queue is not empty, discard its messages, the delete it.

AMPOL_INT_WAIT_INTERVAL

Wait interval. This specifies the time (in milliseconds) to wait when receiving a message using this policy.

Valid values are:

- AMWT_UNLIMITED (unlimited)
- AMINT_PROP_MIN-AMINT_PROP_MAX (that is, 0-99999999)

AMPOL_INT_WAIT_INTERVAL_RO

Wait interval is read-only. This specifies whether or not the wait interval value for this policy is read-only and cannot be changed.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL INT CONVERT

Convert message data on receive. This specifies whether or not to perform code page and encoding conversion on messages received using this policy (target code page and encoding values for the conversion are specified by the selection message).

Valid values are:

- AMPOL NO
- AMPOL YES

AMPOL_INT_WAIT_WHOLE_GROUP

Wait for whole message group. This specifies whether or not to wait for all the messages in a group to become available before returning a message from the group, when receiving a message using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL INT HANDLE POISON MSG

Handle poison messages. This specifies whether or not to perform poison message handling when receiving a message using this policy (see "Note 1" on page 501 for a description of poison message handling).

Valid values are:

AMPOL_NO

AMPOL_YES

AMPOL_INT_ACCEPT_TRUNCATED

Accept truncated messages. This specifies whether or not to accept truncated messages for a message that is too large for the application message buffer when receiving a message using this policy.

Valid values are:

- AMPOL NO
- · AMPOL YES

AMPOL INT SHARED

Open shared. This specifies whether or not to open the underlying message queue as a shared queue (rather than an exclusive queue) when opening a receiver using this policy.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL_INT_FILE_DISP

File disposition. This specifies whether to create a new file, or whether to overwrite or append to an existing file, when receiving a file using this policy.

Valid values are:

- AMPOL_FILE_DISP_NEW
- AMPOL_FILE_DISP_OVERWRITE
- AMPOL_FILE_DISP_APPEND

AMPOL_INT_RETAIN_PUBS

Retain publications. This specifies whether or not the broker retains publications when they are published using this policy.

Valid values are:

AMPOL NO

AMPOL YES

AMPOL_INT_PUB_OTHERS_ONLY

Publish to others only. This specifies whether or not to suppress sending publications back to the publisher (if it is registered as a subscriber) when published using this policy.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL INT SUPPRESS REG

Suppress implicit publisher registration. This specifies whether or not to suppress implicit publisher registration for publications that are sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_PUB_LOCAL

Publish locally. This specifies whether or not publications are sent only to subscribers that are local to the broker when publications are sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL INT ACCEPT DIRECT

Accept direct requests. This specifies whether or not publishers accept direct requests from subscribers when registration results from a publication that is sent using this policy.

Valid values are:

- AMPOL_NO
- · AMPOL YES

AMPOL_INT_PUB_ANON

Publisher registration is anonymous. This specifies whether or not publisher registration is anonymous when registration results from a publication that is sent using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_PUB_USE_CORREL_ID

Use CorrelId as publisher Id. This specifies that the broker uses the CorrelId of the message as part of the publisher's identity when publisher registration results from a publication that is sent using this policy.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL_INT_SUB_LOCAL

Subscribe locally. This specifies whether or not the subscriber is sent only publications published with the local option when it subscribes using this policy.

Valid values are:

- AMPOL_NO
- · AMPOL YES

AMPOL INT NEW PUBS ONLY

Send new publications only. This specifies whether or not a subscriber is sent only new publications when it subscribes using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_PUB_ON_REQ

Publish on request only. This specifies whether or not the subscriber is only sent retained publications on sending a request update when it subscribes using this policy.

Valid values are:

- AMPOL_NO
- AMPOL_YES

AMPOL_INT_INFORM_IF_RET

Inform if retained. This specifies whether or not the broker informs the subscriber whether a publication is retained when it subscribes using this policy.

Valid values are:

- · AMPOL NO
- AMPOL YES

AMPOL_REMOVE_ALL_SUBS

Remove all subscriptions. This specifies whether or not the broker removes all subscriptions for this subscriber when it unsubscribes using this policy.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL_INT_SUB_ANON

Subscriber registration is anonymous. This specifies whether or not subscribers remain anonymous when they subscribe using this policy.

Valid values are:

- AMPOL NO
- AMPOL_YES

AMPOL_INT_SUB_USE_CORREL_ID

Use Correlld as subscriber Id. This specifies that the broker uses the Correlld of the message as part of the subscriber's identity when it subscribes using this policy.

Valid values are:

- AMPOL_NO
- · AMPOL YES

AMPOL INT SUB DLY PERSISTENCE

Delivery persistence of publications. This specifies whether or not publications that the broker sends to subscribers are persistent when they subscribe using this policy.

Valid values are:

- AMPOL_PERSISTENCE_AS_TRP (as transport)
- AMPOL PERSISTENCE NO (not persistent
- AMPOL_PERSISTENCE_YES (persistent)

AMPOL INT USER CONTEXT

User context handle. The AMI does not use this property. It is provided so that other programs, such as a policy handler, can associate their own context information with a policy object.

Valid values are:

AMH_MIN to AMH_MAX

Policy string properties

AMPOL_STR_NAME

The name of the policy object. This provides the function performed by amPolGetName. This property cannot be changed.

AMPOL_STR_APPL_GROUP

Application group. This specifies the application group name when sending to an MQSeries Integrator Version 1 broker using this policy.

AMPOL_STR_SUBS_POINT

Subscription point. This specifies the subscription point when subscribing to an MQSeries Integrator Version 2 broker using this policy.

Service object properties

Service object properties can be integer properties or string properties.

Service integer properties

AMSRV INT TYPE

Service type. This specifies the type of sender or receiver service. This property value is established at object creation time, based on the type of service object being created and its repository definition (if one exists).

Valid values are:

- AMSRV_TYPE_RCV (receiver)
- AMSRV TYPE SND (sender)
- AMSRV_TYPE_DST_SND (distribution list sender)
- AMSRV TYPE PUB SND (publisher sender)
- AMSRV_TYPE_SUB_SND (subscriber sender)
- AMSRV_TYPE_SUB_RCV (subscriber receiver)

AMSRV_INT_SND_TYPE

Sender type. This property is meaningful only for sender services and specifies the sender type. This controls the type and content of Rules and Formatting (RFH) message headers added by the AMI.

Valid values are:

- AMSRV_SND_TYPE_NATIVE (Native default value) Name/value elements are included in an MQRFH header.
- AMSRV_SND_TYPE_MQINT_V1 (MQSeries Integrator V1) Name/value elements are included in an MQRFH header, together with OPT_APP_GROUP and OPT_MSG_TYPE elements, where appropriate.
- AMSRV_SND_TYPE_MQINT_V2 (MQSeries Integrator V2) Name/value elements are included in an MQRFH2 header, together with Default MCD Domain, Default MCD Format, Default MCD Type, Default MCD Format, Delivery Persistence, and Subscription Point elements, where appropriate.
- AMSRV SND TYPE RFH1 (RFH1) Name/value elements are included in an MQRFH header.
- AMSRV SND TYPE RFH2 (RFH2) Name/value elements are included in an MQRFH2 header.

AMSRV_INT_DEFN_TYPE

Definition type. This specifies the definition type and provides the function performed by amRcvGetDefnType or amSubGetDefnType.

This property value is initially set at object creation time to Dynamic, Predefined, or Undefined (the default), based on the repository definition (if one exists). When the service is opened, Undefined or Dynamic may change to Dynamic or Temporary dynamic, based on the attributes of the underlying queue. If a service is used (that is, specified) as the response sender service when receiving a request message, Undefined changes to Response.

Valid values are:

- AMDT_UNDEFINED (Undefined)
- AMDT_RESPONSE (Response)
- AMDT_TEMP_DYNAMIC (Temporary dynamic)
- AMDT_DYNAMIC (Dynamic)
- AMDT_PREDEFINED (Predefined)

AMSRV_INT_ENCODING

Encoding. This provides the function performed by amSndGetEncoding, amPubGetEncoding, or amSubGetEncoding.

This is applicable to sender services with a destination application that cannot perform data conversion for the required encoding. This specifies the encoding of the destination application for this service and exists to provide AMI applications with the information they need to correctly convert message data to the required encoding before it is sent. This property value is set at object creation time based on the repository definition (if one exists).

Valid values are:

- AMENC_NORMAL (Normal)
- AMENC_REVERSED (Reversed)
- AMENC_NORMAL_FLOAT_390 (Normal 390 floating point)
- AMENC REVERSED FLOAT 390 (Reversed, 390 floating point)
- AMENC_UNDEFINED (Undefined default)

AMSRV INT CCSID

Coded character set identifier. This provides the function performed by amSndGetCCSID, amPubGetCCSID, or amSubGetCCSID.

This is applicable to sender services with a destination application that cannot perform data conversion for the required CCSID. This specifies the CCSID of the destination application for this service and provides AMI applications with the information needed to correctly convert message data to the required CCSID before it is sent. This property value is set at object creation time based on the repository definition (if one exists).

Valid values are:

AMINT_PROP_MIN to AMINT_PROP_MAX (that is, 0-99999999)

AMSRV_INT_NEXT_SND_HANDLE

Next sender service handle (in a distribution list). This provides the function performed by amDstGetSenderHandle.

This property value is established from the chain of sender services that represent a distribution list when the distribution list is created. It specifies the handle of the next sender service after this in the distribution list. For the last sender service in a distribution list, this value is set to AMH NULL HANDLE.

Valid values are:

AMH MIN to AMH MAX

AMSRV_INT_SUB_RCV_HANDLE

Subscriber receiver handle. This is applicable only for the sender service of a subscriber.

This property value is established when the subscriber is created and specifies the handle of the receiver service for the subscriber.

Valid values are:

AMH_MIN to AMH_MAX

AMSRV_INT_MQHOBJ

The object handle of the MQSeries queue. This property value is established during AMOP_SENDER_OPEN/MQOPEN or AMOP_RECEIVER_OPEN/MQOPEN processing. A value of AMH_INVALID_HOBJ indicates that the service is closed.

Valid values are:

AMH MIN to AMH MAX

AMSRV_INT_USER_CONTEXT

The user context handle. The AMI does not use this property. It is provided so that other programs, such as a policy handler, can associate their own context information with a service object.

Valid values are:

• AMH_MIN to AMH_MAX

Service string properties

AMSRV STR NAME

The name of the sender or receiver service object. This provides the function performed by amSndGetName, amRcvgetName, amPubGetname, or amSubGetName. This property cannot be changed.

AMSRV STR Q NAME

Queue name. This provides the function performed by amRcvGetQueueName or amSubGetQueueName.

If the definition type is Predefined, this property value is established at object creation time based on the repository definition (if one exists). If the definition type is Dynamic (or Temporary dynamic), it is determined from the name of the underlying dynamic queue that is created when the service is opened. If the definition type is response, it is determined from the name of reply-to queue when receiving a request message.

AMSRV_STR_MGR_NAME

Queue manager name. This is the name of the queue manager where the queue resides. A null string indicates that the queue manager is that to which the AMI session is connected.

If the definition type is Predefined, this property value is established at object creation time based on the repository definition (if one exists). If the definition type is response, it is determined from the name of reply-to queue manager when receiving a request message.

AMSRV_STR_MODEL_Q_NAME

Model queue name. This specifies name of the model queue to be used when opening a service of definition type Dynamic.

AMSRV_STR_DFT_MSG_FORMAT

Default message format. This specifies the default message format to be used if the message format is AMFMT_NONE. It also specifies the value of the AMPS_MSG_TYPE (OPT_MSG_TYPE) element value with sender type MQSeries Integrator V1.

AMSRV_STR_DYNAMIC_PREFIX

Model queue name. This specifies the prefix used for the name of dynamic queue that is created when opening a service of definition type Dynamic.

AMSRV_STR_DFT_MCD_DOMAIN

Default MCD domain. This specifies the AMMCD_MSG_SERVICE_DOMAIN (mcd.Msd) element value with sender type MQSeries Integrator V2.

AMSRV_STR_DFT_MCD_SET

Default MCD set. This specifies the AMMCD_MSG_SERVICE_DOMAIN (mcd.Msd) element value with sender type MQSeries Integrator V2.

AMSRV_STR_DFT_MCD_TYPE

Default MCD type. This specifies the AMMCD_MSG_SERVICE_DOMAIN (mcd.Msd) element value with sender type MQSeries Integrator V2.

AMSRV_STR_DFT_MCD_FORMAT

Default MCD format. This property value is established at object creation time, based on the repository definition (if one exists). It specifies the AMMCD_MSG_SERVICE_DOMAIN (mcd.Msd) element value with sender type MQSeries Integrator V2.

AMSRV_STR_CUST_PARMS

Custom parameters. This specifies the custom parameters defined in the repository for the service point and exists so that a service point can be customized is a way that is meaningful to a policy handler.

Connection object functions

amxConSetStringProp

Sets the specified string property in the connection object. Note that the underlying message transport might ignore or override the values that this function sets.

AMBOOL amxConSetStringProp(
AMHCON hCon,
AMLONG propId,
AMLONG propLen,
AMSTR pProp,
PAMLONG pCompCode,
PAMLONG pReason);

hCon The handle of the connection object that amSesOpen creates

(input).

propId The property identifier (input).

See "Connection string properties" on page 593 for a list of valid

values (you cannot set AMCON_STR_NAME).

propLen The property length (input). If set to AMLEN_NULL_TERM, the

property is a null-terminated string.

pProp The property string (input).
pCompCode Completion code (output).
pReason Reason code (output).

amxConGetStringProp

Returns the specified string property of the connection object.

AMBOOL amxConGetStringProp(
AMHCON hCon,
AMLONG propId,
AMLONG buffLen,
PAMLONG pPropLen,
AMSTR pProp,
PAMLONG pCompCode,
PAMLONG pReason);syntax

hCon The handle of the connection object created during amSesOpen

(input).

propId The property identifier (input).

See "Connection string properties" on page 593 for a list of valid

values.

buffLen The length of the buffer specified by pProp (input). If this is set to

zero, the property string is not returned.

pPropLen The property length excluding any terminating null (output). If this

is set to NULL, the length is not returned.

pProp The property string (output). Any bytes in the buffer after the

property string are set to null, up to the specified buffer length or

property length, whichever is smaller.

pCompCode Completion code (output).

PReason Reason code (output).

amxConSetIntProp

Sets the specified integer property in the connection object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxConSetIntProp(
AMHCON hCon,
AMLONG propId,
AMLONG prop,
PAMLONG pCompCode,
PAMLONG pReason);
```

hCon The handle of the connection object created during amSesOpen

(input).

propId The property identifier (input).

See "Connection integer properties" on page 592 for a list of valid

values.

prop The property value (input).

pCompCode Completion code (output).

pReason Reason code (output).

amxConGetIntProp

Returns the specified integer property of the connection object.

```
AMBOOL amxConGetIntProp(
AMHCON hCon,
AMLONG propId,
PAMLONG pProp,
PAMLONG pCompCode,
PAMLONG pReason);
```

hCon The handle of the connection object created during amSesOpen

(input).

propId The property identifier (input).

See "Connection integer properties" on page 592 for a list of valid

values.

pReason Reason code (output).

Message object functions

amxMsgSetStringProp

Sets the specified string property in the message object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxMsgSetStringProp(
AMHMSG hMsg,
AMLONG propId,
AMLONG propLen,
AMSTR pProp,
PAMLONG pCompCode,
PAMLONG pReason);
```

hMsg The message handle that amSesCreateMessage returns (input).

propId The property identifier (input).

See "Message string properties" on page 602 for a list of valid

values (you cannot set AMMSG_STR_NAME).

propLen The property length (input). If set to AMLEN_NULL_TERM, the

property is a null-terminated string.

pProp The property string (input).
pCompCode Completion code (output).
pReason Reason code (output).

amxMsgGetStringProp

Returns the specified string property of the message object.

```
AMBOOL amxMsgGetStringProp(
AMHMSG hMsg,
AMLONG propId,
AMLONG buffLen,
PAMLONG pPropLen,
AMSTR pProp,
PAMLONG pCompCode,
PAMLONG pReason);
```

hMsg The message handle that amSesCreateMessage returns (input).

propId The property identifier (input).

See "Message string properties" on page 602 for a list of valid

values.

buffLen The length of the buffer specified by pProp (input). If this is set to

zero, the property string is not returned.

pPropLen The property length excluding any terminating null (output). If this

is set to NULL, the length is not returned.

pProp The property string (output). Any bytes in the buffer after the

property string are set to null, up to the specified buffer length or

property length, whichever is smaller.

pCompCode Completion code (output).

pReason Reason code (output).

amxMsgSetIntProp

Sets the specified integer property in the message object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxMsgSetIntProp(
AMHMSG hMsg,
AMLONG propId,
AMLONG prop,
PAMLONG pCompCode,
PAMLONG pReason);
```

hMsg The message handle that amSesCreateMessage returns (input).

propId The property identifier (input).

See "Message integer properties" on page 595 for a list of valid

values.

prop The property value (input).

pCompCode Completion code (output).

pReason Reason code (output).

amxMsgGetIntProp

Returns the specified integer property of the message object.

```
AMBOOL amxMsgGetIntProp(
AMHMSG hMsg,
AMLONG propId,
PAMLONG pProp,
PAMLONG pCompCode,
PAMLONG pReason);
```

hMsg The message handle that amSesCreateMessage returns (input).

propId The property identifier (input).

See "Message integer properties" on page 595 for a list of valid

values.

prop The property value (output).

pCompCode Completion code (output).

Reason code (output).

amxMsgAssemble

Assembles a message in wire format with all the required headers (MQRFH message element headers and external headers), ready for transmission. The message data can be provided either by the message object, or from an external data buffer. An optional external message header can be inserted at the start of message. An additional option allows a message header to be inserted at the start of a previously assembled message.

This function enables a policy handler to replace the existing MQSeries message transport. It is called from a message transport routine before a send function.

This function returns the address of the buffer containing the newly assembled wire format message. The length is determined using amxMsgGetIntProps with property identifier AMMSG_INT_MSG_LEN.

AMBOOL amxMsgAssemble(AMHMSG hMsg, AMLONG options, AMLONG extDataLen, PAMBYTE pExtData, AMLONG extHdrLen, PAMBYTE pExtHdr, PPAMBYTE pBufferAddr, PAMLONG pCompCode, PAMLONG pReason);

hMsg The message handle that amSesCreateMessage returns (input).

options Options (input). Valid values are:

AMMSG_AMO_NONE

Assemble the message in the default way.

AMMSG_AMO_RFH

Include the message elements in an MQRFH header.

AMMSG_AMO_RFH2

Include the message elements in an MQRFH2 header.

AMMSG_AMO_EXT_HDR

Prefix an external header, but do not change the assembled

message.

extDataLen Length of data in external buffer (input).

If the data is contained in the message object and there is no

external data buffer, set this to zero.

pExtData Pointer to data in an external buffer (input).

If the data is contained in the message object and there is no

external data buffer, set this to NULL.

extHdrLen External header length (input).

If there is no external header, set this to zero.

pExtHdr Pointer to an external header to add to the start of the message

(input).

If there is no external header, set this to NULL.

pBufferAddr Pointer to the assembled message data (output).

pCompCode Completion code (output).

pReason Reason code (output).

amxMsgAllocateMem

Allocates space in the message object when receiving a message directly into the message object, rather than into an external buffer.

This function enables a policy handler to replace the existing MQSeries message transport. It is called from a message transport routine before a receive function.

```
AMBOOL amxMsgAllocateMem(
AMHMSG hMsg,
AMLONG minLen,
PAMLONG pBuffLen,
PPAMBYTE pBuffAddr,
PAMLONG pCompCode,
PAMLONG pReason);
```

hMsg The message handle that amSesCreateMessage returns (input).

minLen The minimum required length in bytes (input). A value of zero

specifies the default size.

pBuffLen Length allocated in bytes (output).pBuffAddr Address of allocated buffer (output).

pCompCode Completion code (output).

pReason Reason code (output).

amxMsgUpdated

Parses a newly received message so that any RFH name/value elements can be extracted from the message data into an internal element table. The message data can be contained in the message object, or in an external data buffer.

This function enables a policy handler to replace the existing MQSeries message transport. It is called from a message transport routine after a message is received and all required message properties are updated.

```
AMBOOL amxMsgUpdated(
AMHMSG hMsg,
AMLONG extMsgLen,
PAMBYTE pExtMsg,
PAMLONG pCompCode,
PAMLONG pReason);
```

hMsg The message handle that amSesCreateMessage returns (input).

extMsgLen The length of the message data in an external buffer property

identifier (input).

If the data is received into the message object and there is no

external data buffer, set this to zero.

pExtMsg Pointer to the message data in an external buffer property

identifier (input).

If the data is received into the message object and there is no

external data buffer, set this to NULL.

pCompCode Completion code (output).

pReason Reason code (output).

Policy object functions

amxPolSetStringProp

Sets the specified string property in the policy object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxPolSetStringProp(
AMHPOL hPolicy,
AMLONG propId,
AMLONG propLen,
AMSTR pProp,
PAMLONG pCompCode,
PAMLONG pReason);
```

hPolicy The policy handle that amSesCreatePolicy returns (input).

propId The property identifier (input).

See "Policy string properties" on page 610 for a list of valid values

(you cannot set AMPOL_STR_NAME).

propLen The property length (input). If set to AMLEN_NULL_TERM, the

property is a null-terminated string.

pProp The property string (input).
pCompCode Completion code (output).
pReason Reason code (output).

amxPolGetStringProp

Returns the specified string property of the policy object.

```
AMBOOL amxPolGetStringProp(
AMHPOL hPolicy,
AMLONG propId,
AMLONG buffLen,
PAMLONG pPropLen,
AMSTR pProp,
PAMLONG pCompCode,
PAMLONG pReason);
```

hPolicy The policy handle that amSesCreatePolicy returns (input).

propId The property identifier (input).

See "Policy string properties" on page 610 for a list of valid values.

buffLen The length of the buffer specified by pProp (input). If this is set to

zero, the property string is not returned.

pPropLen The property length excluding any terminating null (output). If this

is set to NULL, the length is not returned.

pProp The property string (output). Any bytes in the buffer after the

property string are set to null, up to the specified buffer length or

property length, whichever is smaller.

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

amxPolSetIntProp

Sets the specified integer property in the policy object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxPolSetIntProp(
AMHPOL hPolicy,
AMLONG propId,
AMLONG prop,
PAMLONG pCompCode,
PAMLONG pReason);
```

hPolicy The policy handle that amSesCreatePolicy returns (input).

propId The property identifier (input).

See "Policy integer properties" on page 603 for a list of valid

values.

prop The property value (input).

pCompCode Completion code (output).

pReason Reason code (output).

amxPolGetIntProp

Returns the specified integer property of the policy object.

```
AMBOOL amxPolGetIntProp(
AMHPOL hPolicy,
AMLONG propId,
PAMLONG pProp,
PAMLONG pCompCode,
PAMLONG pReason);
```

hPolicy The policy handle that amSesCreatePolicy returns (input).

propId The property identifier (input).

See "Policy integer properties" on page 603 for a list of valid

values.

pProp The property value (output).

pCompCode Completion code (output).

pReason Reason code (output).

Service object functions

amxSrvSetStringProp

Sets the specified string property in the service object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxSrvSetStringProp(
AMHSRV hSrv,
AMLONG propId,
AMLONG propLen,
AMSTR pProp,
PAMLONG pCompCode,
PAMLONG pReason);
```

hSrv The service handle that amSesCreateSender, amSesCreateReceiver,

amSesCreatePublisher, or amSesCreateSubscriber returns (input).

propId The property identifier (input).

See "Service string properties" on page 613 for a list of valid values

(you cannot set AMSRV_STR_NAME).

propLen The property length (input). If set to AMLEN_NULL_TERM, the

property is a null-terminated string.

pProp The property string (input).
pCompCode Completion code (output).

pReason Reason code (output).

amxSrvGetStringProp

Returns the specified string property of the service object.

```
AMBOOL amxSrvGetStringProp(
AMHSRV hSrv,
AMLONG propId,
AMLONG buffLen,
PAMLONG pPropLen,
AMSTR pProp,
PAMLONG pCompCode,
PAMLONG pReason);
```

hSrv The service handle that amSesCreateSender, amSesCreateReceiver,

amSesCreatePublisher, or amSesCreateSubscriber returns (input).

propId The property identifier (input).

See "Service string properties" on page 613 for a list of valid

values.

buffLen The length of the buffer specified by pProp (input). If this is set to

zero, the property string is not returned.

pPropLen The property length excluding any terminating null (output). If this

is set to NULL, the length is not returned.

pProp The property string (output). Any bytes in the buffer after the

property string are set to null, up to the specified buffer length or

property length, whichever is smaller.

pCompCode Completion code (output).

pReason Reason code (output).

Extended C AMI functions

amxSrvSetIntProp

Sets the specified integer property in the service object. Note that the underlying message transport might ignore or override the values that this function sets.

```
AMBOOL amxSrvSetIntProp(
AMHSRV hSrv,
AMLONG propId,
AMLONG prop,
PAMLONG pCompCode,
PAMLONG pReason);
```

hSrv The service handle that amSesCreateSender, amSesCreateReceiver,

amSesCreatePublisher, or amSesCreateSubscriber returns (input).

propId The property identifier (input).

See "Service integer properties" on page 611 for a list of valid

values.

prop The property value (input).

pCompCode Completion code (output).

pReason Reason code (output).

amxSrvGetIntProp

Returns the specified integer property of the service object.

```
AMBOOL amxSrvGetIntProp(
AMHSRV hSrv,
AMLONG propId,
PAMLONG pProp,
PAMLONG pCompCode,
PAMLONG pReason);
```

hSrv The service handle that amSesCreateSender, amSesCreateReceiver,

amSesCreatePublisher, or amSesCreateSubscriber returns (input).

propId The property identifier (input).

See "Service integer properties" on page 611 for a list of valid

values.

pProp The property value (output).
pCompCode Completion code (output).
pReason Reason code (output).

Appendix D. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation North Castle Drive Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing 2-31 Roppongi 3-chome, Minato-ku Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

The following permission statements and conditions only apply to the XML library files (AMTXML, amtxml310, libamtxml310, and libamtxml310_r) built using software from Apache Software Foundation, and not to any code developed solely by IBM Corporation. They do not invalidate any of the terms of the IBM International Program License Agreement for this IBM product.

Copyright © 1999-2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.
- 4. The names "Xerces" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache\@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Trademarks

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

AS/400 AIX BookManager CICS DB2 **IBM IBMLink** Language Environment **MQSeries** OS/390 **RACF** OS/400

VSE/ESA VisualAge

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.

Notices

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.

A

ADSI. Active Directory Service Interfaces.

B

broker. See message broker.

C

COM. Component Object Model.

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*.

Customer Information Control System (CICS). An IBM licensed program that provides online transaction processing services and management for critical business applications.

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*.

Directory Information Tree (DIT). The complete set of information to which the directory provides access, considered as a tree, whose vertices (other than the root) are the directory entries.

distinguished name (DN). One of the names of an object, formed from the sequence of Relative Distinguished Names (RDNs) of its object entry and each of its superior entries.

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

E

Encina. A product that supplies transaction management services.

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that was derived from, and is a subset of, SGML. 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*.

G

Generic Security Service API (GSS API). An application programming interface enabling application programs that do not implement remote procedure calls (RPCs) to have security services provided by a server in a Distributed Computing Environment (DCE). The GSS API provides security services to callers through a generic method that functions independently of underlying cryptography mechanisms or communication protocols and can thus be used in many different environments. The GSS API became available as part of the Open Software Foundation's (OSF's) Release 1.1 of DCE.

Н

HP. Hewlett-Packard Company.

ILE. Integrated Language Environment (IBM OS/400).

Information Management System (IMS). A data base/data communication system capable of managing complex data bases and terminal networks.

Internet Engineering Task Force (IETF). The task force of the Internet Architecture Board (IAB) that is responsible for solving the short-term engineering needs of the Internet and that develops new Internet standard specifications.

Glossary

J

job control language (JCL). A control language used to identify a job to an operating system and to describe the job's requirements.

JNI. Java Native Interface.

Lightweight Directory Access Protocol (LDAP). An open industry standard that defines a protocol for the requests and responses that flow between directory clients and servers.

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).

Ν

NDS. Novell Directory Services.

0

OAMAS. Open Applications Group Middleware Application Programming Interface Specification.

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 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.

recoverable resource services (RRS). An OS/390 facility that provides two-phase syncpoint support across participating resource managers.

relational database manager (RDBM). For DB2, the database backend that preceded SDBM and TDBM.

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.

RRS. recoverable resource services.

S

SASL. Simple Authentication and Security Layer.

Secure Sockets Layer (SSL). A security protocol that provides communication privacy. SSL enables client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery.

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.

SSL. Secure Sockets Layer.

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.

Τ

TCP/IP. Transmission Control Protocol/Internet Protocol. A set of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

TDBM. For DB2, the database backend that followed RDBM and SDBM.

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

Transport Layer Security (TLS). A security protocol defined by the Internet Engineering Task Force (IETF) that is based on Secure Sockets Layer (SSL) and is specified in RFC 2246.

U

URL. Uniform Resource Locator. A sequence of characters that represent information resources on a computer or in a network such as the Internet.

W

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

X

XML. Extensible Markup Language.

Glossary

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 and Windows 2000 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/mgseries/

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 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

Α	AmDistributionList (C++) (continued)	AmErrorException (C++) (continued)
	overview 189	getSource 236
Accept Direct Requests attribute	3C110 ZZ1	overview 195
Accept Truncated Message attribut	Scharific 221	toString 236
Active Directory 506	AmDistributionList (Java)	AmErrorException (Java)
disabling the schema 509	clearErrorCodes 420	getClassName 433
schema installation 508	close 420	getCompletionCode 433
addElement	enableWarnings 420	getMethodName 433
AmMessage (C++) 205	getLastErrorStatus 420	getReasonCode 433
AmMessage (Java) 405	getName 420	getSource 433
addFilter	getSender 420	overview 397
AmMessage (C++) 205	getSenderCount 420	toString 433
AmMessage (Java) 406	open 420	AmException (C++)
addTopic	overview 392	getClassName 235
AmMessage (C++) 205	send 421	getCompletionCode 235
AmMessage (Java) 406	sendFile 421	getMethodName 235
administration tool	AMDLCL 348	getReasonCode 235
installation 477	AMDLCLEC 348	getSource 235
objects and attributes 491	AMDLGTLE 348	overview 195
policy handlers 149	AMDLGTNA 349	toString 235
updating LDAP directories 50	9 AMDLGTSC 349	using 171
admqami.ldf 508	AMDLGTSH 350	AmException (Java)
admqami.vbs 508	AMDLOP 350	getClassName 432
AIX	AMDLSN 351	getCompletionCode 432
C++ applications 175	AMDLSNFL 351	getMethodName 432
C applications 30	amDstClearErrorCodes 131	getReasonCode 432
installation 444	amDstClose 131	getSource 432
Java applications 382	amDstGetLastError 132	overview 397
prerequisites 441	amDstGetName 132	toString 432
schema installation, SecureWay	amDstGetSenderCount 133	using 379
Directory 507	amDstGetSenderHandle 133	AMFB constants 563
AMB constants 561	amDstOpen 133	AMFMT constants 563
amBackout 44	amDstSend 134	AMGF constants 563
amBegin 45	amDstSendFile 135	AMGRP constants 563
amBrowseMsg 46	AMDT constants 561	AMH constants 563, 571
AMBRW constants 561	AMEI constants 571	AMHBACK 263
AmBytes (C++)	AMELEM structure 24, 253, 562	AMHBEGIN 264
cmp 228	AmElement (C++)	AMHBRMS 265
constructors 228	constructor 230	AMHCMIT 267
cpy 229	getName 230	AMHINIT 268
dataPtr 229	getValue 230	AMHPB 269
destructor 229	getVersion 230	AMHRCFL 270
length 229	overview 193	AMHRCMS 272
operators 229	setVersion 230	AMHRCPB 274
overview 193	toString 230	AMHRCRQ 276
pad 229	using 171	AMHSB 282
AMCC constants 561	AmElement (Java)	AMHSNFL 278
amCommit 48	constructor 429	AMHSNMS 279
AMCON constants 571	getName 429	AMHSNRQ 280
AMCON_INT constants 571	getValue 429	AMHSNRS 281
AMCON_STR constants 571	getVersion 429	AMHTERM 283
AmConstants (Java) 428	overview 396	AMHUN 284
AMDEF constants 561	setVersion 429	AMI C 580
AmDistributionList (C++)	toString 429	AMI C begin parameter structure 577
clearErrorCodes 220	using 378	AMI C close connection parameter
close 220	AMENC constants 562	structure 577
enableWarnings 220	AmErrorException (C++)	AMI C close distribution list parameter
getLastErrorStatus 220	getClassName 236	structure 578
getName 220	getCompletionCode 236	AMI C close service parameter
getSender 220	getMethodName 236	structure 578
getSenderCount 220	getReasonCode 236	AMI C commit parameter structure 578
open 220		

AMI C element structure 562	AmMessage (C++) (continued)	AmMessage (Java) (continued)
AMI C handle poison message parameter	getElementCount 207	setGroupStatus 411
structure 579	getEncoding 207	setReportCode 411
AMI C MQBACK parameter	getFilter 207	setType 412
structure 583	getFilterCount 207	writeBytes 412
AMI C MQBEGIN parameter	getFormat 207	AMMSADEL 314
	O	AMMSADFI 314
structure 583	getGroupStatus 208	
AMI C MQCLOSE parameter	getLastErrorStatus 208	AMMSADTO 315
structure 584	getMessageId 208	AMMSCLEC 315
AMI C MQCMIT parameter	getName 208	AMMSDEEL 315
structure 584	getNamedElement 208	AMMSDEFI 316
AMI C MQCONN parameter	getNamedElementCount 208	AMMSDENE 316
structure 585	getReportCode 209	AMMSDETO 317
AMI C MQCONNX parameter	getTopic 209	AMMSG constants 572
structure 585	getTopicCount 209	AMMSG_INT constants 573
AMI C MQDISC parameter	getType 209	AMMSG_STR constants 573
structure 586	overview 185	amMsgAddElement 95
AMI C MQGET parameter structure 586	readBytes 209	amMsgAddFilter 95
<u>*</u>		9
AMI C MQINQ parameter structure 587	reset 209	AmMsgAddStreamName 113
AMI C MQOPEN parameter	setCCSID 210	amMsgAddTopic 96
structure 587	setCorrelationId 210	amMsgClearErrorCodes 96
AMI C MQPUT parameter structure 588	setDataOffset 210	amMsgDeleteElement 96
AMI C MQPUT1 parameter	setElementCCSID 210	amMsgDeleteFilter 97
structure 588	setEncoding 210	amMsgDeleteNamedElement 97
AMI C MQSET parameter structure 589	setFormat 211	amMsgDeleteTopic 98
AMI C open connection parameter	setGroupStatus 211	AMMSGELC 317
structure 579	setReportCode 211	amMsgGetCCSID 98
AMI C open distribution list parameter	setType 212	amMsgGetCorrelId 98
structure 579	writeBytes 212	amMsgGetDataLength 99
	•	9
AMI C open service parameter	AmMessage (Java)	amMsgGetDataOffset 99
structure 580	addElement 405	amMsgGetElement 99
AMI C receive from service parameter	addFilter 406	amMsgGetElementCCSID 100
structure 581	addTopic 406	amMsgGetElementCount 100
AMI C rollback parameter structure 581	clearErrorCodes 406	amMsgGetEncoding 100
AMI C send to distribution list parameter	deleteElement 406	amMsgGetFilter 101
structure 582	deleteFilter 406	amMsgGetFilterCount 101
AMI C send to service parameter	deleteNamedElement 406	amMsgGetFormat 102
structure 582	deleteTopic 407	amMsgGetGroupStatus 102
AMI extensions, policy handlers 157	enableWarnings 407	amMsgGetLastError 103
AMI operation code constants 573	getCCSID 407	amMsgGetMsgId 103
AMI operation invocation points, policy	getCorrelationId 407	amMsgGetName 104
	9	9
handler 150	getDataLength 407	amMsgGetNamedElement 104
AMI parameter union 580	getDataOffset 407	amMsgGetNamedElementCount 105
amInitialize 49	getElement 407	AmMsgGetPubTimeStamp 113
AMINT constants 571	getElementCount 407	amMsgGetReportCode 105
AMINV constants 571	getEncoding 408	AmMsgGetStreamName 114
AMLEN constants 563	getFilter 408	amMsgGetTopic 106
amLibRegisterFunction 157	getFilterCount 408	amMsgGetTopicCount 107
amLibTraceText 157	getFormat 408	amMsgGetType 106
AMLONG constants 572	getGroupStatus 408	amMsgReadBytes 107
AMMCD constants 563	getLastErrorStatus 408	amMsgReset 107
AmMessage (C++)	getMessageId 409	amMsgSetCCSID 108
addElement 205	getName 409	amMsgSetCorrelId 108
	ĕ	amMsgSetCoffend 100 amMsgSetDataOffset 108
addFilter 205	getNamedElement 409	0
addTopic 205	getNamedElementCount 409	amMsgSetElementCCSID 109
clearErrorCodes 205	getReportCode 409	amMsgSetEncoding 109
deleteElement 205	getTopic 409	amMsgSetFormat 110
deleteFilter 205	getTopicCount 409	amMsgSetGroupStatus 110
deleteNamedElement 206	getType 410	amMsgSetReportCode 111
deleteTopic 206	overview 388	amMsgSetType 111
enableWarnings 206	readBytes 410	AMMSGTCC 317
getCCSID 206	reset 410	AMMSGTCI 318
getCorrelationId 206	setCCSID 410	AMMSGTDL 318
getDataLength 206	setCorrelationId 410	AMMSGTDO 318
getDataDefigiti 200 getDataOffset 206	setDataOffset 410	AMMSGTEC 319
0		
getElement 207	setEncoding 411	AMMSGTEL 319
getElementCCSID 207	setFormat 411	AMMSGTEN 320

AMMSGTFC 320	AMPHMQSET structure 589	AmPublisher (Java) (continued)
AMMSGTFI 321	AMPHOPC structure 579	open 423
AMMSGTFO 321	AMPHOPD structure 579	overview 393
AMMSGTGS 322	AMPHOPS structure 580	publish 423
AMMSGTLE 322	AMPHPARM structure 580	amPubOpen 138
AMMSGTMI 323	AMPHRBK structure 581	amPubPublish 139
AMMSGTNA 323	AMPHRCS structure 581	AMRC constants 564
AMMSGTNC 324	AMPHSND structure 582	AMRCBR 338
AMMSGTNE 324	AMPHSNS structure 582	AMRCBRSE 340
AMMSGTRC 325	AMPOCLEC 364	AMRCCL 342
AMMSGTTC 326	AMPOGTLE 364	AMRCCLEC 341
AMMSGTTO 325	AMPOGTNA 365	AMRCGTDT 342
AMMSGTTY 326	AMPOGTWT 365	AMRCGTLE 343
amMsgWriteBytes 112	AMPOINTER constant 563	AMRCGTNA 343
AMMSREBY 327	AMPOL constants 574	AMRCGTQN 344
AMMSRS 327	AMPOL_INT constants 574	AMRCOP 344
AMMSSELC 328	AMPOL_STR constants 575	AMRCRC 345
AMMSSTCC 328	amPolClearErrorCodes 147	AMRCRCFL 346
AMMSSTCI 328	amPolGetLastError 147	AMRCSTQN 347
AMMSSTDO 329	amPolGetName 148	amRcvBrowse 120
AMMSSTEN 329	amPolGetWaitTime 148	amRcvBrowseSelect 122
AMMSSTFO 330	AmPolicy (C++)	amRcvClearErrorCodes 123
AMMSSTGS 330	clearErrorCodes 227	amRcvClose 124
AMMSWRBY 331	enableWarnings 227	amRcvGetDefnType 124
AMMT constants 563	getLastErrorStatus 227	amRcvGetLastError 125
AmObject (C++)	getName 227	amRcvGetName 125
clearErrorCodes 231	getWaitTime 227	amRcvGetQueueName 126
getLastErrorStatus 231	overview 192	amRcvOpen 126
getName 231	setWaitTime 227	amRcvReceive 127
overview 193	AmPolicy (Java)	amRcvReceiveFile 129
AmObject (Java)	clearErrorCodes 427	amRcvSetQueueName 130
clearErrorCodes 430	enableWarnings 427	amReceiveFile 51
getLastErrorStatus 430 getName 430	getLastErrorStatus 427 getName 427	amReceiveMsg 53 amReceivePublication 55
overview 396	getWaitTime 427	AmReceiver (C++)
AMOP constants 573	overview 395	browse 216
AMPBCL 353	setWaitTime 427	clearErrorCodes 217
AMPBCLEC 353	amPolSetWaitTime 148	close 217
AMPBGTCC 353	AMPOSTWT 366	enableWarnings 217
AMPBGTEN 354	AMPROP constants 575	getDefinitionType 217
AMPBGTLE 354	AMPS constants 563	getLastErrorStatus 218
AMPBGTNA 355	amPubClearErrorCodes 136	getName 218
AMPBOP 355	amPubClose 136	getQueueName 218
AMPBPB 356	amPubGetCCSID 136	open 218
AMPH constants 577	amPubGetEncoding 137	overview 188
AMPHBGN structure 577	amPubGetLastError 137	receive 218
AMPHCLC structure 577	amPubGetName 138	receiveFile 219, 419
AMPHCLD structure 578	amPublish 50	setQueueName 219
AMPHCLS structure 578	AmPublisher (C++)	AmReceiver (Java)
AMPHCMT structure 578	clearErrorCodes 222	browse 416
AMPHHPM structure 579	close 222	clearErrorCodes 417
amPhlCreate 152	enableWarnings 222	close 417
amPhlDelete 152	getCCSID 222	enableWarnings 417
amPhlInitialize 152	getEncoding 222	getDefinitionType 417
amPhlXxx 153	getLastErrorStatus 222	getLastErrorStatus 418
AMPHMQBACK structure 583	getName 222	getName 418
AMPHMQBEGIN structure 583	open 223	getQueueName 418
AMPHMQCLOSE structure 584	overview 190	open 418
AMPHMQCMIT structure 584	publish 223	overview 391
AMPHMQCONNY structure 585	AmPublisher (Java)	receive 418
AMPHMQCONNX structure 585	clearErrorCodes 422	setQueueName 419
AMPHMODISC structure 586	close 422	amReceiveRequest 57
AMPHMQGET structure 586	enableWarnings 422	AMRO constants 577
AMPHMQINQ structure 587 AMPHMQOPEN structure 587	getCCSID 422 getEncoding 422	AMSBCL 357 AMSBCLEC 357
AMPHMQOPEN structure 587 AMPHMQPUT structure 588	getLastErrorStatus 422	AMSBGTCC 358
	genasimioisialus 422	1 11 1 1 D D D D D D D D D D D D D D D
AMPHMQPUT1 structure 588	getName 422	AMSBGTDT 358

AMSBGTEN 359	AMSERB 312	AmSession (Java) (continued)
AMSBGTLE 359	amSesBegin 82	createPolicy 403
AMSBGTNA 360	amSesClearErrorCodes 82	createPublisher 403
AMSBGTQN 360	amSesClose 83	createReceiver 403
AMSBOP 361	amSesCommit 83	createSender 403
AMSBRC 361	amSesCreate 83	createSubscriber 403
AMSBSB 362	amSesCreateDistList 84	enableWarnings 404
AMSBSTQN 362	amSesCreateMessage 84	getLastErrorStatus 404
AMSBUN 363	amSesCreatePolicy 84	getName 404
AMSD constants 570	amSesCreatePublisher 85	getTraceLevel 404
AMSEBG 300	amSesCreateReceiver 85	getTraceLocation 404
AMSECL 301	amSesCreateSender 86	open 404
AMSECLEC 300	amSesCreateSubscriber 86	overview 387
AMSECM 301	amSesDelete 87	rollback 404
AMSECR 302	amSesDeleteDistList 87	transaction coordination 402
AMSECRDL 302	amSesDeleteMessage 87	unit of work 402
AMSECRMS 303	amSesDeletePolicy 88	AmSessionFactory (C++)
AMSECRPB 304	amSesDeletePublisher 88	constructors 198
	amSesDeleteReceiver 88	createSession 198
AMSECRPO 303		
AMSECRRC 304	amSesDeleteSender 89	deleteSession 198
AMSECRSB 305	amSesDeleteSubscriber 89	getFactoryName 198
AMSECRSN 305	amSesGetDistListHandle 89	getLocalHost 198
AMSEDL 306	amSesGetLastError 90	getRepository 198
AMSEDLDL 306	amSesGetMessageHandle 90	getTraceLevel 198
AMSEDLMS 306	amSesGetPolicyHandle 91	getTraceLocation 198
AMSEDLPB 307	amSesGetPublisherHandle 91	overview 182
AMSEDLPO 307	amSesGetReceiverHandle 91	setLocalHost 199
AMSEDLRC 307	amSesGetSenderHandle 92	setRepository 199
AMSEDLSB 308	amSesGetSubscriberHandle 92	setTraceLevel 199
AMSEDLSN 308	amSesOpen 92	setTraceLocation 199
AMSEGHDL 308	amSesRollback 93	AmSessionFactory (Java)
AMSEGHMS 309	AmSession (C++)	constructor 400
AMSEGHPB 310	begin 200	createSession 400
AMSEGHPO 310	clearErrorCodes 200	getFactoryName 400
AMSEGHRC 310	close 200	getLocalHost 400
AMSEGHSB 311	commit 200	getRepository 400
AMSEGHSN 311	createDistributionList 200	getTraceLevel 400
AMSEGTLE 309	createMessage 201	getTraceLocation 400
AmSender (C++)	createPolicy 201	overview 386
clearErrorCodes 213	createPublisher 201	setLocalHost 400
close 213	createReceiver 201	setRepository 401
enableWarnings 213	createSender 201	setTraceLevel 401
getCCSID 213	createSubscriber 201	setTraceLocation 401
getEncoding 213	deleteDistributionList 202	AMSNCL 333
getLastErrorStatus 214 getName 214	deleteMessage 202 deletePolicy 202	AMSNCLEC 332 amSndClearErrorCodes 115
	deletePublisher 202	amSndClose 115
open 214		
overview 187	deleteReceiver 202	amSndGetCCSID 116
send 214	deleteSender 202	amSndGetEncoding 116
sendFile 215	deleteSubscriber 202	amSndGetLastError 116
AmSender (Java)	enableWarnings 202	amSndGetName 117
clearErrorCodes 413	getLastErrorStatus 203	amSndOpen 117
close 413	getName 203	amSndSend 118
enableWarnings 413	getTraceLevel 203	amSndSendFile 119
getCCSID 413	getTraceLocation 203	AMSNGTCC 333
getEncoding 413	open 203	AMSNGTEN 333
getLastErrorStatus 414	overview 183	AMSNGTLE 334
getName 414	rollback 203	AMSNGTNA 334
open 414	transaction coordination 200	AMSNOP 335
overview 390	unit of work 200	AMSNSN 335
send 414	AmSession (Java)	AMSNSNFL 336
sendFile 415	begin 402	AMSRV constants 575
amSendFile 59	clearErrorCodes 402	AMSRV_INT constants 576
amSendMsg 60	close 402	AMSRV_STR constants 576
amSendRequest 61	commit 402	AmStatus (C++)
amSendResponse 62	createDistributionList 402	constructor 232
AMSEOP 312	createMessage 403	getCompletionCode 232
	U	0 1

AmStatus (C++) (continued)	AmSubscriber (Java) (continued)	amxSrvSetIntProp 624
getReasonCode 232	open 425	amxSrvSetStringProp 623
getReasonCode2 232	overview 394	Anonymous Registration attribute 502,
overview 193	receive 425	503
toString 232	setQueueName 425	appearance of text in this book xxii
using 171	subscribe 426	Application Group attribute 498
AmStatus (Java)	unsubscribe 426	application messaging interface
constructor 431	amSubSetQueueName 144	basic model 7
getCompletionCode 431	amSubSubscribe 145	description 4
getReasonCode 431	amSubUnsubscribe 146	interoperability 3
getReasonCode2 431	AMT_DATA_PATH environment	introduction 3
overview 396	variable 471	main components 3
toString 431	AMT_HOST environment variable 471	programming languages 4
using 379	AMT_LDAP_AUTHENTICATION	receiving messages 3
AMSTR constants 576	environment variable 513	sending messages 3
AmString (C++)	AMT_LDAP_BINDDN environment	applications, building
cat 233	variable 512	C 29
cmp 233	AMT_LDAP_PASSWORD environment	C++ 175
constructors 233	variable 513	COBOL 257
contains 233	AMT_REPOSITORY environment	Java 382
cpy 233	variable 471, 474, 512	applications, writing
destructor 233	AMT_SSL environment variable 513	C 16
length 234	AMTC constants 576	C++ 165
operators 234	amtc.h header 29	COBOL 246
overview 194	AMTELEMV and AMTELEML	Java 373
pad 234	copybooks 253	AS/400
split 234	amTerminate 64	C++ applications 176
strip 234	amtsphlr sample program 487	C applications 31
text 234	amtsw.ldif 507	installation 448
truncate 234	amtsw390.at 508	Java applications 382
amSubClearErrorCodes 140	amtsw390.ldif 507	prerequisites 441
amSubClose 140	amtsw390.oc 508	sample programs 481
amSubGetCCSID 140	amUnsubscribe 65	assemble message option constants 572
amSubGetEngoding 141	AmWarningException (C++)	attributes, policy
amSubGetEncoding 141 amSubGetLastError 142	getClassName 237 getCompletionCode 237	general 498 handler 503
amSubGetName 142	getMethodName 237	initialization 497
amSubGetQueueName 143	getReasonCode 237	publish 503
amSubOpen 143	getSource 237	receive 500
amSubReceive 144	overview 195	send 498
amSubscribe 63	toString 237	subscribe 502
AmSubscriber (C++)	AmWarningException (Java)	attributes, policy handler 504
clearErrorCodes 224	getClassName 434	Available Service Points attribute 496
close 224	getCompletionCode 434	
enableWarnings 224	getMethodName 434	
getCCSID 224	getReasonCode 434	В
getDefinitionType 224	getSource 434	_
getEncoding 224	overview 397	base classes
getLastErrorStatus 224	toString 434	C++ 163, 197
getName 225	AMWT constants 570	Java 371, 399
getQueueName 225	amxConGetIntProp 616	begin
open 225	amxConGetStringProp 615	AmSession (C++) 200
overview 191	amxConSetIntProp 616	AmSession (Java) 402
receive 225	amxConSetStringProp 615	bibliography 633
setQueueName 225	amxMsgAllocateMem 620	Bind On Open attribute 498
subscribe 226	amxMsgAssemble 618	BookManager 635 Boolean constants 561
unsubscribe 226	amxMsgGetIntProp 618	Broker Service attribute 496
AmSubscriber (Java)	amxMsgGetStringProp 617	browse
clearErrorCodes 424	amxMsgSetIntProp 618	AmReceiver (C++) 216
close 424	amxMsgSetStringProp 617	AmReceiver (Java) 416
enableWarnings 424	amxMsgUpdated 620	constants 561
getCCSID 424	amxPolGetIntProp 622	building applications
getDefinitionType 424	amxPolGetStringProp 621	C 29
getEncoding 424	amxPolSetIntProp 622	C++ 175
getLastErrorStatus 424	amxPolSetStringProp 621	COBOL 257
getName 425	amxSrvGetIntProp 624	Java 382
getQueueName 425	amxSrvGetStringProp 623	,a 50 <u>-</u>

C	close	constants (continued)
	AmDistributionList (C++) 220	group status 563
C++ applications AIX 175	AmDistributionList (Java) 420	handle 563
AS/400 176	AmPublisher (C++) 222	handle interval 571
HP-UX 177	AmPublisher (Java) 422	handle property limit 571
Solaris 178	AmReceiver (C++) 217	integer property true/false 575
Windows 179	AmReceiver (Java) 417	invocation point 571
C++ interface	AmSender (C++) 213 AmSender (Java) 413	maximum string length 576 Message Content Descriptor tag
overview 181	AmSession (C++) 200	names 563
reference information 197	AmSession (Java) 402	message object integer property
using 163	AmSubscriber (C++) 224	identifiers 573
C applications	AmSubscriber (Java) 424	message object integer property
AIX 30	closing a session	value 572, 575
AS/400 31	C 16	message object string property
HP-UX 32	C++ 165	identifiers 573
OS/390 33	COBOL 246	message types 563
Solaris 34	Java 373	pointer definition 563
Windows 35	COBOL applications	policy handler continuation code 577
C high-level interface	OS/390 257	policy handler transport type 577
equivalent object interface functions 78	COBOL high-level interface	policy object integer property
overview 42	equivalent object interface	identifiers 574
reference information 43	functions 296	policy object integer property
using 13	overview 260	value 574
C object interface	reference information 262	policy object string property identifiers 575
extended functions 591	using 243 COBOL object interface	positive integer property limit 571
overview 67	overview 285	publish/subscribe 563
reference information 81	reference information 299	reason codes 564
using 13	using 243	receive options 577
C policy handler interface	commit	service and policy definitions 561
AMI extensions 157	AmSession (C++) 200	service object integer property
invocation points 150	AmSession (Java) 402	identifier 576
library functions 151	compilers 442	service object string property
overview 149	compiling a policy handler 36	identifier 576
cache, repository and local host	completion code constants 561	signed integer property limit 572
(OS/390) 475	connecting to MQSeries 478	string length 563
callback requests, policy handlers 157 CCSID attribute 494	Connection Mode attribute 497	system default names and
class (C++)	Connection Name attribute 497	handles 570
base 163, 197	connection object	trace control 576
exception 164, 197	functions 615	wait time 570
helper 164, 197	integer properties 592	content-based publish/subscribe 479 Convert attribute 500
class (Java)	string properties 593 connection object integer property	createDistributionList
base 371, 399	identifiers 571	AmSession (C++) 200
exception 372, 399	connection object integer property value	AmSession (Java) 402
helper 372, 399	constants 571	createMessage
clearErrorCodes	connection object string property	AmSession (C++) 201
AmDistributionList (C++) 220	identifiers 571	AmSession (Java) 403
AmDistributionList (Java) 420	Connection Type attribute 497	createPolicy
AmMessage (C++) 205	constants	AmSession (C++) 201
AmMessage (Java) 406	AMI operation code 573	AmSession (Java) 403
AmObject (C++) 231 AmObject (Java) 430	assemble message option 572	createPublisher
AmPolicy (C++) 227	Boolean 561	AmSession (C++) 201
AmPolicy (Java) 427	browse 561	AmSession (Java) 403
AmPublisher (C++) 222	completion codes 561	createReceiver
AmPublisher (Java) 422	connection object integer property identifier 571	AmSession (C++) 201 AmSession (Java) 403
AmReceiver (C++) 217	connection object integer property	AmSession (Java) 403 createSender
AmReceiver (Java) 417	value constants 571	AmSession (C++) 201
AmSender (C++) 213	connection object string property	AmSession (Java) 403
AmSender (Java) 413	identifier 571	createSession 403
AmSession (C++) 200	definition type 561	AmSessionFactory (C++) 198
AmSession (Java) 402	encoding 562	AmSessionFactory (Java) 400
AmSubscriber (C++) 224	expiry interval 571	createSubscriber
AmSubscriber (Java) 424	feedback codes 563	AmSession (C++) 201
Client Channel Name attribute 497	format 563	AmSession (Java) 403
Client TCP Server Address attribute 497		

creating MySeries objects 489 directory search (LDAF 512 directory server (Creating objects) (Creating objec			
Custom Parameters attribute 494	creating MQSeries objects 480	directory search, LDAP 512	encoding constants 562
Custom Parameters attribute 494 D Active Directory 506 directory structure AIX 445 AS/400 449 HP-LX 455 CS/500 459 Solaris 463 AS/400 489 HP-LX 455 CS/500 459 Solaris 463 CP-1 166 COBOL 246 Directory Update vindow AIX 447 AS/400 489 Directory Update vindow AIX 447 AS/400 489 Solaris 463 Directory Update vindow AIX 445 AS/400 489 Solaris 463 Directory Update vindow AIX 445 AS/400 489 Solaris 463 Solaris 464 COBOL 246 Défault MCD Domain attribute 494 Octavities MCD Domain attribute 494 COEMIL MCD Type attribute 494 COEMIL MCD Type attribute 494 COEMIL 404 COEMIL 244 Java 372 aystem 492 definition dispers 494 default objects C 14 C++ 164 COBOL 244 Java 372 aystem 492 definition Type attribute 494 default objects AB ITI-UX 455 CC 16 CC ++ 171 COBOL 255 Java 373 Exception Action attribute 494 COEMIL 244 Java 372 ambStcGerlad Cant 133 amDStCGen 132 ambStCGen 134 ambStcGerlad 134 ambS		*	
SecureWay Directory 506 Custom Parameters attribute 494			AIX 445
D		,	
ANX 445			
AS-/400 449	Custom Furameters attribute 474		
D			
data types, C. 29 datagram C+ 166 COBOL 246 Liva 373 Default Format attribute 494 Default MCD Domain attribute 494 Default MCD Set attrib	D		0 0
Solaris 462 Solaris 462 Amilons Amilon	ט		
datagram C 16 C++ 166 C++ 166 C++ 166 C++ 166 C++ 166 C++ 166 C-+ 166 Java 373 Default MCD Domain attribute 494 Default MCD Type attribute 494 Default MCD	data types, C 29		· · · · · · · · · · · · · · · · · · ·
C 16 C+1 166 COBOL 246 COBOL 247 Default MCD Domain attribute 494 Default MCD Domain attribute 494 Default MCD Domain attribute 494 Default MCD Format attribute 494 Default MCD Format attribute 494 Default MCD Format attribute 494 Default MCD Type attribute 496 Default MCD Type	* 1		
C++ 166 COBOL 246 Java 373 Default Pormat attribute 494 Default MCD Format attribute 494 Default MCD Format attribute 494 Default MCD Set attribute 494 Default MCD Set attribute 494 Default MCD Type attribute 494 Default MCD Set attribute 494 Default MCD Type attribute 494 Default MCD Set attribute 494 Default Objects C 14 C++ 164 CCDBOL 292 overview (Gv-+) 189 Default Objects C 18 COBOL 246 Java 372 ambs:Cedes Before Count 133 ambs:CleastError 132 a		Windows 468	Windows 467
Administration fool 509		Directory Update window	error handling
Default MCD Domain attribute 494 Default MCD Format attribute 494 Default MCD Set		AMI Administration Tool 509	C 25
Default NCD Domain attribute 494 Default MCD Format attribute 494 Default MCD Format attribute 494 Default MCD Sype attribute 494 Default MCD Type attribute 496 Default MCD Type attribut		disk space 441	C++ 171
Default MCD Format attribute 494 Default MCD Format attribute 494 Default MCD Format attribute 494 Default MCD Type attribute 494 COBOL 244	•	distribution list definition 496	COBOL 255
Default MCD Format attribute 494 Default MCD Format attribute 494 Default MCD Type attribute 494 default objects C 14 C++ 164 COBOL 244 Java 372 system 492 definition distribution list 496 policy 491, 497 policy handler 491, 504 publisher 496 service 491 subscriber 496 service 491 subscriber 496 system provided 492 Definition Type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteDistribution list AmSession (C++) 202 deleteEletement AmMessage (C++) 205 AmMessage (G++) 205 AmMessage (G++) 205 AmMessage (G++) 206 AmMessage (G++) 202 deleteNamedIllement AmMessage (C++) 202 deleteVamedIllement AmMessage (C++) 202 Ammession (C++) 202 deleteVamedIllement Ammessage (C++) 202 Ammession (C++) 202 deleteVamedIllement Ammession (C++) 202 deleteVamedIllement Ammession (C++) 202 deleteVamedIllement Ammession (C++) 202 Ammession (C++) 202 deleteVamedIllement Ammession (C++) 202 deleteVamedIllement Ammession (C++) 202 deleteVamedIllement Ammession (C++) 202 deleteVamedIllement Ammession (C++) 202 Ammession (C++) 202 deleteVamedIllement Ammession (C++) 202		distribution list interface	Java 379
Default MCD Format attribute 494 Default MCD Type attribute 494 Default MCD Type attribute 494 Default MCD Type attribute 494 default objects C 14 C++ 164 COBOL 244 Java 372 system 492 definition distribution list 496 policy 491, 497 policy handler 491, 504 publisher 496 service 911 service point (sender/receiver) subscriber 496 system provided 492 Definition type attribute 494 definition type constants 561 Definition type attribute 500 deleteDistributionList Ambessage (C++) 205 AmMessage (G++) 205 AmMessage (G++) 205 AmMessage (G++) 205 AmMessage (G++) 206 AmMessage (C++) 202 deleteNameDefinent AmMessage (C++) 202 deleteNameDefine 133 ambsKcestender-flandle 13		overview (C) 74	
Default MCD Type attribute 494 default objects C 14 C++ 164 COBOL 244 Java 372 system 492 default objects C 14 COBOL 244 Java 372 system 492 definition distribution list 496 policy 491, 497 policy 491, 497 policy 491, 497 policy 491, 907 policy 491, 497 policy 491, 497			1
Default MCD Type attribute 494	Default MCD Set attribute 494	· · ·	
C 14	Default MCD Type attribute 494		
C+1 164	default objects	- ·	
COBOL 244 Java 372 amDstClose 131 amDstGetLastError 132 amDstGetLastError 132 amDstGetLastError 132 amDstGetLastError 132 amDstGetLastError 132 amDstGetSenderCount 133 amDstGetSenderCount 133 amDstGetSenderCount 133 amDstGetSenderCount 133 amDstGetSenderFinalde 133 amDstGetSenderFinalde 133 amDstGetSenderFinalde 133 amDstGetSenderFinalde 133 amDstGetSenderFinalde 135 amDstGetSenderFinalde 136 amDstGetSenderFinalde 136 amDstGetSenderFinalde 137 amDstGetSenderFinalde 133 amDstGetSenderGetCOBOL 2	C 14	. ,	
Java 372 system 492 definition distribution list 496 policy 491, 497 policy handler 491, 504 publisher 496 service 491 service point (sender/receiver) subscriber 496 system provided 492 Definition Type attribute 494 definition type constants 561 Definition Type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteEbirenent AmMessage (C++) 202 AmMessage (C++) 205 AmMessage (Ana) 406 deleteEhitenent AmMessage (C++) 205 AmMessage (Ana) 406 deletePolicy AmMessage (Ana) 406 deletePolicy AmMessage (Ana) 406 deletePolicy AmMessage (Ana) 406 deleteReceiver Ammession (C++) 202 deleteRec	C++ 164		1
Java 372 system 492 definition distribution list 496 policy 491, 497 policy handler 491, 504 publisher 496 service 491 service point (sender/receiver) subscriber 496 system provided 492 Definition Type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteDistributionList Ambession (C++) 202 deleteEnlern AmMessage (C++) 205 AmMessage (Java) 406 deleteManedElement AmMessage (C++) 202 deleteNamedElement AmMessage (C++) 202 deletePolicy Ambession (C++) 202 deletePolicy Ambession (C++) 202 deletePolicy Ambession (C++) 202 deleteReceiver Amsession (COBOL 244		-
amDst.ectvame 132 amDst.ectscame 132 amDst.ectscame 132 amDst.ectscame 132 amDst.ectscame 132 amDst.ectscame 133 amDst.ectscame 132 amDst.ectscame 134 amDst.ectscame 135 amDst.ectscame 134 amDst.ectscame 132 amDst.ectscame 134 amDst.ectscame			
definition distribution list 496 policy 491, 497 policy handler 491, 504 publisher 496 service 491 service point (sender/receiver) system provided 492 Definition Type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteDistributionList Ambessage (C++) 202 deleteElement AmMessage (C++) 205 AmMessage (Ama) 406 deleteFilter AmMessage (C++) 205 AmMessage (Java) 406 deleteNamedElement AmMessage (C++) 202 deleteNamedElement AmMessage (C++) 202 deletePublisher AmMessage (C++) 202 deletePublisher AmMessage (C++) 202 deleteReceiver AmSession (C++) 202 deleteSender AmSession (C++) 202 deleteReceiver AmSession (C++) 202 deleteSender AmSession (C++) 202 deleteS			
distribution list 496 policy 491, 497 policy handler 491, 504 publisher 496 service 491 service point (sender/receiver) 494 subscriber 496 system provided 492 Definition Type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteDistributionList AmSession (C++) 202 deleteElement AmMessage (C++) 205 AmMessage (G++) 206 AmMessage (C++) 202 deleteReceiver AmSession (C++) 202 deleteReceiver AmSession (C++) 202 deleteReceiver AmSession (C++) 202 deleteSender			
policy 491, 497 policy handler 491, 504 publisher 496 service 491 service point (sender/receiver) 494 subscriber 496 system provided 492 Definition Type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteDistributionList AmSession (C++) 202 deleteEment AmMessage (Java) 406 deleteMessage AmSession (C++) 205 AmMessage (Java) 406 deleteMessage AmSession (C++) 202 deleteMessage AmMessage (C++) 205 AmMessage (Java) 406 deletePolicy AmMessage (C++) 206 AmMessage (Java) 406 deletePolicy AmSession (C++) 202 deleteMessage AmSession (C++) 202 deletePolicy AmSession (C++) 202 deleteSeasion AmSession (C++) 202 deleteSeasion AmSession (C++) 202 deleteSeasion (C++) 203 deleteSeasion (C++) 204 deleteSeasion (C++) 205 deleteSeasion (C++) 206 deleteSeasion (C++) 207 deleteSeasion (C++) 208 deleteSeasion (C++) 209 deleteSeasion (C++) 209 deleteSeasion (C++) 201 deleteSeasion (C++) 202 deleteSeasion (C++) 202 deleteSeasion (C++) 203 deleteSeasion (C++) 204 deleteSeasion (C++) 205 deleteSeasion (C++) 206 deleteSeasion (C++) 207 deleteSeasion (C++) 208 deleteSeasion (C++) 209 deleteSeasion (C++) 209 deleteSeasion (C++) 201 deleteSeasion (C++) 202 deleteSeasion (C++) 203 deleteSea		amDstGetSenderHandle 133	expiry interval constant 571
policy handler 491, 504 publisher 496 service 491 service point (sender/receiver) subscriber 496 system provided 492 Definition type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteDistributionList AmMessage (C++) 202 deleteElement AmMessage (C++) 205 AmMessage (G++) 206 AmMessage (G++) 202 deleteReceiver AmSession (C++) 202 deleteSesion AmSession (amDstOpen 133	
publisher 496 service 491 service point (sender/receiver) subscriber 496 system provided 492 pefinition Type attribute definition Type attribute def	1 7	amDstSend 134	extensions, policy handlers 157
service 491 service with sender/receiver) 494 subscriber 496 system provided 492 Definition Type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteEllement AmMessage (C++) 205 AmMessage (Java) 406 deleteFilter AmMessage (Java) 406 deleteMessage (Java) 406 deleteNamedEllement AmMessage (Java) 406 deleteVamedEllement AmMessage (Java) 406 deleteVamedEllement AmMessage (Java) 406 deletePolicy AmSession (C++) 202 deleteSage AmSession (C++) 202 deleteSage (Java) 406 deletePolicy AmSession (C++) 202 deleteSage (Java) 406 deleteSage (Java) 406 deletePolicy AmSession (C++) 202 deleteSage (Java) 406 deletePolicy AmSession (C++) 202 deleteSage (Java) 406 deleteSage (Java) 407		amDstSendFile 135	
service 491 service point (sender/receiver) 494 subscriber 496 system provided 492 Definition Type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteDistributionList AmSession (C++) 202 AmMessage (C++) 205 AmMessage (C++) 206 deleteNamedElement AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (G++) 206 AmMessage (G++) 202 deletePolicy AmSession (C++) 202 deletePublisher AmSession (C++) 202 deleteReceiver AmSession (C++) 202 deleteReceiver AmSession (C++) 202 deleteReceiver AmSession (C++) 202 deleteSesson AmSession (C++) 202 deleteSesson AmSession (C++) 202 deleteSession AmSession (C++) 202 deleteSubscriber AmSession (C++) 202	1	distribution list interface (COBOL)	
subscriber 496 system provided 492 pefinition type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteDestributionList Ammol.GTIS 349 AMDLGTSC 349 AMDLGTSC 349 AMDLGTSC 349 AMDLGTSC 349 AMDLGTSC 349 AMDLSN 351 Dynamic Queue Prefix attribute 494 deleteElement AmMessage (C++) 205 AmMessage (G++) 205 AmMessage (G++) 205 AmMessage (G++) 202 C++ 171 COBOL 253 AMDLSN 351 Field Disposition attribute 500 field limits C 28 C++ 174 COBOL 256 Java 381 fields AMI C 580 AMI C 680		AMDLCL 348	F
subscriber 496 system provided 492 Definition Type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteDistributionList AmSession (C++) 202 deleteElment AmMessage (Java) 406 deleteFilter AmMessage (Java) 406 deleteNessage AmSession (C++) 202 deletePublisher AmMessage (Java) 406 deletePublisher AmMessage (Java) 406 deletePublisher AmSession (C++) 202 AmDistributionList (Java) 420 AmSession (C++) 202 deleteReceiver AmSession (C++) 202 AmMessage (Java) 406 deletePublisher AmSession (C++) 202 deleteReceiver AmSession (C++) 202 AmSession (C++) 202 deleteReceiver AmSession (C++) 202 deleteReceiver AmSession (C++) 202 deleteSession AmSession (C++) 202 deleteSubscriber AmSession (C++) 202 deleteSubscriber AmSession (C++) 202 AmSession (C++)		AMDLCLEC 348	
Definition Type attribute 494 definition type constants 561 Delete On Close attribute 500 deleteDistributionList AmSession (C++) 202 deleteElement AmMessage (C++) 205 AmMessage (G++) 206 AmMessage (G++) 202 AmMessage (G++) 202 AmMessage (G++) 206 AmMessage (G++) 206 AmMessage (G++) 206 AmMessage (G++) 206 AmMessage (G++) 207 AmSession (C++) 208 AmSession (C++) 209 AmMessage (Gava) 407 AmSession (C++) 209 AmSession (C++) 209 AmMessage (Gava) 407 AmSes		AMDLGTLE 348	
Definition type constants 561 AMDLGTSC 349 AMDLCTSH 350 AMDLDN 350 AMDLSN 351 AMDLSNSTL 351 AMDLSNSage (C++) 202 AMDLSNSage (G++) 205 AMMSsage (Java) 406 deleteFilter AmMessage (Java) 406 deleteMessage AmSession (C++) 202 AmMessage (Java) 406 deletePublisher AmMessage (Java) 406 deletePublisher AmSession (C++) 202 AmSession (C++) 202 deleteReceiver AmSession (C++) 202 deleteSession AmSession (C++) 202 AmPublisher (Java) 427 deleteSubscriber AmSession (C++) 202 AmReceiver (C++) 217 AmSession (C++) 202 AmSession (C++) 202 AmSession (C++) 202 AmReceiver (Java) 417 AmSession (C++) 202 AmMessage (Java) 407 AmSession (C++) 202 AmMessage (Java) 407 AmSession (C++) 202 AmMessage (Java) 407 AmSession (C++) 202 AmSession (C++) 202 AmMessage (Java) 407 AmSession (C++) 202 AmSessio			
Delete On Close attribute 500 deleteDistributionList AmSession (C++) 202 deleteElement AmMessage (C++) 205 AmMessage (Dava) 406 deleteMessage (Dava) 406 deleteMessage (Dava) 406 deleteMessage (Dava) 406 deleteNessage (Dava) 407 AmMessage (Dava) 407 AmMessage (Dava) 407 AmSession (C++) 202 deleteNession (C++) 202 deleteNession (C++) 202 deleteNession (Dava) 407 AmSession (Dava) 407 AmSes			
Delete On Close attribute AmSession (C++) 202 deleteElement AmMessage (C++) 205 AmMessage (C++) 206 AmSession (C++) 202 AmSession (C++) 206 AmMessage (C++) 206 AmSession (C++) 202 AmSession (C++) 203 AmSession (C++) 204 AmSession (C++) 205 AmSession (C++) 206 AmSession (C++) 207 AmSession (C++) 208 AmSession (C++) 209 AmSession (C++) 201 AmSession (C++) 202 AmSession (C++) 203 AmSession (C++) 204 AmSession (C++) 205 AmSession (C++) 206 AmSession (C++) 207 AmSession (C++) 208 AmSession (C++) 209 AmSession (C++) 201 AmSession (C++) 202 AmSession (C++) 203 AmSession (C++) 204 AmSession (C++) 205 AmMessage (G++) 206 AmMe	definition type constants 561		
Ambessage (C++) 205 Ambessage (C++) 205 Ambessage (C++) 205 Ambessage (C++) 205 Ambessage (G++) 202 Ambessage (G++) 202 Ambessage (G++) 202 Ambessage (G++) 203 Ambessage (G++) 204 Ambessage (G++) 205 Ambessage (G++) 206 Ambessage (G++) 207 Ambessage (G++) 208 Ambessage (G++) 208 Ambessage (G++) 209 Ambessage (G++) 206 Ambess	Delete On Close attribute 500		symptom report (UNIX and
Ambession (C++) 202 Ambession (C++) 205 Amkessage (G++) 206 Amkessage (C++) 206 Amkessage (C++) 206 Amkessage (G++) 202 Ambession (C++) 202 Amkessage (G++) 206 Amkessage (G++) 206 Amkessage (G++) 206 Amkessage (G++) 206 Amkessage (G++) 207 Ambession (C++) 208 Ambession (C++) 209 Ambess	deleteDistributionList		Windows) 532
AmMessage (C++) 205 AmMessage (G++) 202 AmMessage (G++) 202 AmSession (C++) 202 AmMessage (C++) 204 AmMessage (C++) 205 AmMessage (G++) 206 AmMessage (G++) 207 AmMessage (G++) 208 AmMessage (G++) 208 AmMessage (G++) 208 AmMessage (G++) 208 AmMessage (G++) 209 AmMessage (G++) 200 AmDistributionList (C++) 220 AmDistributionList (G++) 220 AmDistributionList (G++) 200 AmMessage (G++) 206 AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (C++) 207 AmSession (C++) 208 AmMessage (G++) 209 AmMessage (G++) 209 AmPolicy (G++) 227 AmSession (C++) 209 AmPolicy (G++) 227 AmI C commit parameter structure 578 AmI C deleteSession AmReceiver (G++) 217 AmI C MQBACK parameter AmSession (C++) 202 AmSender (G++) 213 AmI C MQBACK parameter AmSession (C++) 202 AmMessage (G++) 206 AmSession (G++) 202 AmSender (G++) 202 AmMessage (G++) 206 AmSession (G++) 202 AmSender (G++) 202 AmMessage (G++) 206 AmSession (G++) 202 AmSender (G++) 202 AmMessage (G++) 206 AmSender (G++) 202 AmSender (G++) 202 AmSender (G++) 202 AmMessage (G++) 206 AmSender (G++) 202 AmMessage (G++) 206 AmSender (G++) 202 AmSender (G++) 202 AmI C MORDISTIONALISTIONALISTIONALISTIONALISTIONALI	AmSession (C++) 202		feedback codes 563
AmMessage (Java) 406 deleteFilter AmMessage (C++) 205 AmMessage (G++) 205 AmMessage (Java) 406 deleteMessage (C++) 205 AmMessage (Java) 406 deleteMessage (C++) 202 deleteMessage (C++) 202 deleteNamedElement AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (Java) 406 deletePolicy AmMessage (Java) 406 deletePolicy AmSession (C++) 202 AmDistributionList (C++) 220 AmDistributionList (C++) 220 AmMessage (Java) 407 AmSession (C++) 202 AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 208 AmMessage (C++) 209 AmMessage (C++) 227 AmSession (C++) 202 AmPolicy (C++) 227 AmSession (C++) 202 AmPolicy (Gava) 427 AmSession (C++) 202 AmPublisher (Java) 427 AmSession (C++) 202 AmPublisher (Java) 422 AmMessage (Java) 407 AmSession (C++) 217 AmSession (C++) 208 AmReceiver (Java) 417 AmSession (C++) 208 AmMessage (Java) 407 AmSession (C++) 208 AmSession (C++) 209 AmMessage (Java) 407 AmSubscriber (Java) 413 AMI C MQBACK parameter structure 583 AMI C MQCLOSE parameter structure 584	deleteElement		Field Disposition attribute 500
AmMessage (Java) 406 C 28 C + 174 C C + 174 C C C C C C C C C	AmMessage (C++) 205	Dynamic Queue Frenx attribute 494	field limits
deleteFilter AmMessage (C++) 205 AmMessage (Java) 406 deleteMessage AmSession (C++) 202 deleteNamedElement AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (Java) 406 deletePolicy AmMessage (Java) 406 deletePolicy AmSession (C++) 202 AmDistributionList (C++) 220 AmDistributionList (Java) 420 AMI C close connection parameter structure 577 AmSession (C++) 202 AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (C++) 206 AmSession (C++) 202 AmMessage (C++) 206 AmMessage (C++) 206 AmMessage (C++) 207 AmSession (C++) 202 AmMessage (Java) 407 AMI C close distribution list parameter structure 578 AmI C close distribution list parameter structure 578 AmI C close exprice parameter structure 578 AmI C close service parameter structure 578 AmI C close service parameter deleteReceiver AmPolicy (C++) 227 AmMessage (Java) 407 AmPolicy (Java) 427 AMI C commit parameter structure 578 AmI C commit parameter structure 578 AmI C commit parameter structure 578 AmI C close service parameter deleteSender AmPublisher (C++) 222 AmPublisher (C++) 222 AmPolicy (Java) 427 AMI C commit parameter structure 579 AmI C amid poison message Amid C mid C			C 28
AmMessage (C++) 205 AmMessage (Java) 406 deleteMessage AmSession (C++) 202 deleteNamedElement AmMessage (C++) 206 AmMessage (D++) 207 AmSession (C++) 208 AmMessage (D++) 209 AmPublisher (D++) 222 AmPublisher (D++) 222 AmPublisher (D++) 222 AmPublisher (D++) 222 AmPublisher (D++) 213 AmMi C element 562 AmReceiver (C++) 217 AmMessage (D++) 208 AmReceiver (D++) 213 Ami C MogBACK parameter AmSession (C++) 202 AmSession (C++) 202 AmSender (D++) 213 Ami C MogBACK parameter AmSession (C++) 202 AmSession (C++) 202 AmSession (D++) 202 AmMessage (D++) 206 AmMessage (D++) 206 AmMessage (D++) 206 AmSession (D++) 204 Ami C MogBack parameter AmSession (D++) 205 Ami C MogBack parameter Ami C MogCLOSE parameter Ami C MogCLOSE parameter	0 - ,	_	C++ 174
AmMessage (Java) 406 elements, name/value Glete Message C 24 fields AmSession (C++) 202 C++ 171 AMI C 580 deleteNamedElement COBOL 253 AMI C begin parameter AmMessage (C++) 206 Java 378 structure 577 AmMessage (Java) 406 enableWarnings AMI C close connection parameter deletePolicy AmDistributionList (C++) 220 structure 577 AmSession (C++) 202 AmDistributionList (Java) 420 AMI C close distribution list deletePublisher AmMessage (C++) 206 parameter structure 578 AmSession (C++) 202 AmMessage (Java) 407 AMI C close service parameter deleteReceiver AmPolicy (C++) 227 structure 578 AmSession (C++) 202 AmPolicy (Java) 427 AMI C commit parameter deleteSender AmPublisher (Java) 422 structure 578 AmSession (C++) 202 AmPublisher (Java) 422 AMI C element 562 deleteSession AmReceiver (C++) 217 AMI C lement 562 deleteSession AmReceiver (Java) 417 parameter structure 579 deleteSubscriber AmSession (C++) 202 AmSender (Java) 417 parameter structure 579 deleteSubscriber AmSender (C++) 213 AMI C MQBACK parameter AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmSession (C++) 202 AmSender (Java) 413 deleteTopic AmSession (C++) 204 AmSession (Java) 404 structure 583 deleting C++ objects 166 AmSubscriber (Java) 424 structure 584	AmMessage (C++) 205	E	COBOL 256
deleteMessage C 24 fields AmSession (C++) 202		elements name/value	
AmSession (C++) 202 C++ 171 deleteNamedElement COBOL 253 AmMessage (C++) 206 AmMessage (G++) 206 AmMessage (G++) 206 AmMessage (G++) 206 AmMessage (G++) 202 AmDistributionList (C++) 220 AmSession (C++) 202 AmDistributionList (G++) 206 AmSession (C++) 202 AmMessage (G++) 206 AmMessage (G++) 206 AmMessage (G++) 206 AmSession (C++) 202 AmMessage (G++) 206 AmSession (C++) 202 AmMessage (G++) 227 AmSession (C++) 202 AmPolicy (G++) 227 AmSession (C++) 202 AmPublisher (G++) 222 AmPublisher (G++) 222 AmRession (C++) 202 AmRession (C++) 202 AmRession (C++) 217 AmGession (C++) 202 AmRession (C++) 217 AmI C lose service parameter structure 578 AmI C commit parameter structure 578 AmI C commit parameter structure 578 AmI C close service parameter structure 578 AmI C commit parameter structure 578 AmI C commit parameter structure 578 AmI C close service parameter structure 578 AmI C commit parameter structure 578 AmI C commit parameter structure 578 AmI C close service parameter structure 578 AmI C close service parameter structure 578 AmI C commit parameter structure 578 AmI C close service parameter structure 578 AmI C deleteSession structure 578 AmI C deleteSession structure 578 AmI C deleteSession structure 579 AmI C deleteSession struc			
deleteNamedElementCOBOL 253AMI C begin parameterAmMessage (C++) 206Java 378structure 577AmMessage (Java) 406enableWarningsAMI C close connection parameterdeletePolicyAmDistributionList (C++) 220structure 577AmSession (C++) 202AmDistributionList (Java) 420AMI C close distribution listdeletePublisherAmMessage (C++) 206parameter structure 578AmSession (C++) 202AmMessage (Java) 407AMI C close service parameterdeleteReceiverAmPolicy (C++) 227structure 578AmSession (C++) 202AmPolicy (Java) 427AMI C commit parameterdeleteSenderAmPublisher (C++) 222structure 578AmSession (C++) 202AmPublisher (Java) 422AMI C element 562deleteSessionAmReceiver (C++) 217AMI C handle poison messageAmSessionFactory (C++) 198AmReceiver (Java) 417parameter structure 579deleteSubscriberAmSender (C++) 213AMI C MQBACK parameterAmSession (C++) 202AmSender (Java) 413structure 583deleteTopicAmSession (C++) 202AMI C MQBEGIN parameterAmMessage (C++) 206AmSession (Java) 404structure 583AmMessage (Java) 407AmSubscriber (C++) 224AMI C MQCLOSE parameterdeleting C++ objects 166AmSubscriber (Java) 424structure 584	O		
AmMessage (C++) 206	, ,		
AmMessage (Java) 406 enableWarnings deletePolicy AmDistributionList (C++) 220 structure 577 AmSession (C++) 202 AmDistributionList (Java) 420 AMI C close distribution list deletePublisher AmMessage (C++) 206 parameter structure 578 AmSession (C++) 202 AmMessage (Java) 407 AmSession (C++) 202 AmMessage (Java) 427 AmSession (C++) 202 AmPolicy (Java) 427 AmSession (C++) 202 AmPublisher (C++) 222 deleteSender AmSession (C++) 202 AmPublisher (Java) 422 deleteSession AmSession (C++) 202 AmPublisher (Java) 422 deleteSession AmSession (C++) 198 AmReceiver (Java) 417 deleteSubscriber AmSession (C++) 202 AmSender (Java) 417 deleteSubscriber AmSession (C++) 202 AmSender (Java) 413 deleteTopic AmSession (C++) 202 AmSender (Java) 413 deleteTopic AmMessage (Java) 407 AmSession (Java) 404 AmSubscriber (C++) 224 AmI C MQBEGIN parameter AmMessage (Java) 407 AmSubscriber (Java) 424 AmI C MQCLOSE parameter structure 583 AmI C MQCLOSE parameter structure 584			
deletePolicy AmSession (C++) 202 AmDistributionList (C++) 220 AMI C close distribution list deletePublisher AmSession (C++) 202 AmMessage (C++) 206 AmSession (C++) 202 AmMessage (Java) 407 AMI C close service parameter structure 578 AMI C close service parameter deleteReceiver AmPolicy (C++) 227 AmSession (C++) 202 AmPolicy (Java) 427 AMI C commit parameter structure 578 AmSession (C++) 202 AmPublisher (C++) 222 AmPublisher (Java) 422 AMI C element 562 deleteSession AmSession (C++) 198 AmReceiver (C++) 217 AMI C handle poison message AmSession (C++) 198 AmReceiver (Java) 417 AmSender (C++) 213 AMI C MQBACK parameter AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmSession (C++) 202 AmSession (Java) 404 AmSession (Java) 424 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 AMI C MQCLOSE parameter	9 1 1	*	
AmSession (C++) 202 AmDistributionList (Java) 420 AMI C close distribution list deletePublisher AmMessage (C++) 206 parameter structure 578 AmSession (C++) 202 AmMessage (Java) 407 AMI C close service parameter deleteReceiver AmPolicy (C++) 227 structure 578 AmSession (C++) 202 AmPolicy (Java) 427 AMI C commit parameter deleteSender AmPublisher (C++) 222 structure 578 AmSession (C++) 202 AmPublisher (Java) 422 AMI C element 562 deleteSession AmReceiver (C++) 217 AMI C handle poison message AmSessionFactory (C++) 198 AmReceiver (Java) 417 parameter structure 579 deleteSubscriber AmSender (C++) 213 AMI C MQBACK parameter AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmSession (C++) 202 AMI C MQBEGIN parameter AmMessage (C++) 206 AmSession (Java) 404 structure 583 AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584			
deletePublisherAmMessage (C++)206parameter structure578AmSession (C++)202AmMessage (Java)407AMI C close service parameterdeleteReceiverAmPolicy (C++)227structure578AmSession (C++)202AmPolicy (Java)427AMI C commit parameterdeleteSenderAmPublisher (C++)222structure578AmSession (C++)202AmPublisher (Java)422AMI C element562deleteSessionAmReceiver (C++)217AMI C handle poison messageAmSessionFactory (C++)198AmReceiver (Java)417parameter structure579deleteSubscriberAmSender (C++)213AMI C MQBACK parameterAmSession (C++)202AmSender (Java)413structure583deleteTopicAmSession (Java)404structure583AmMessage (Java)407AmSession (Java)404structure583AmMessage (Java)407AmSubscriber (C++)224AMI C MQCLOSE parameterdeleting C++ objects166AmSubscriber (Java)424structure584		` /	
AmSession (C++) 202 AmMessage (Java) 407 AMI C close service parameter deleteReceiver AmPolicy (C++) 227 structure 578 AmSession (C++) 202 AmPolicy (Java) 427 AMI C commit parameter deleteSender AmPublisher (C++) 222 structure 578 AmSession (C++) 202 AmPublisher (Java) 422 AMI C element 562 deleteSession AmReceiver (C++) 217 AMI C handle poison message AmSessionFactory (C++) 198 AmReceiver (Java) 417 parameter structure 579 deleteSubscriber AmSender (C++) 213 AMI C MQBACK parameter AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmSession (C++) 202 AMI C MQBEGIN parameter AmMessage (C++) 206 AmSession (Java) 404 structure 583 AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584	* *	- · · · · · · · · · · · · · · · · · · ·	
deleteReceiver AmPolicy (C++) 227 AmSession (C++) 202 AmPolicy (Java) 427 AmI C commit parameter AmPublisher (C++) 222 AmI C element 562 AmI C handle poison message AmSessionFactory (C++) 198 AmReceiver (Java) 417 AmI C handle poison message AmSessionFactory (C++) 213 AmI C MQBACK parameter AmSession (C++) 202 AmSender (Java) 413 AmI C MQBACK parameter AmSession (C++) 202 AmSession (C++) 202 AmI C MQBEGIN parameter AmMessage (C++) 206 AmSession (Java) 404 AmSession (Java) 404 AmSubscriber (C++) 224 AmI C MQBEGIN parameter AmMessage (Java) 407 AmSubscriber (C++) 224 AmI C MQCLOSE parameter		9 1 1	
AmSession (C++) 202 AmPolicy (Java) 427 AMI C commit parameter deleteSender AmPublisher (C++) 222 structure 578 AmSession (C++) 202 AmPublisher (Java) 422 AMI C element 562 AmReceiver (C++) 217 AMI C handle poison message AmSessionFactory (C++) 198 AmReceiver (Java) 417 parameter structure 579 deleteSubscriber AmSession (C++) 202 AmSender (Java) 413 structure 583 AmI C MQBACK parameter AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmSession (C++) 202 AmI C MQBEGIN parameter AmMessage (C++) 206 AmSession (Java) 404 structure 583 AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584	AmSession (C++) 202	AmMessage (Java) 407	AMI C close service parameter
deleteSender AmPublisher (C++) 222 structure 578 AmSession (C++) 202 AmPublisher (Java) 422 AMI C element 562 AMI C handle poison message AmSessionFactory (C++) 198 AmReceiver (Java) 417 parameter structure 579 deleteSubscriber AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmMessage (C++) 206 AmSession (Java) 404 structure 583 AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQBEGIN parameter AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584	deleteReceiver	AmPolicy (C++) 227	structure 578
AmSession (C++) 202 AmPublisher (Java) 422 AMI C element 562 deleteSession AmReceiver (C++) 217 AMI C handle poison message AmSessionFactory (C++) 198 AmReceiver (Java) 417 parameter structure 579 deleteSubscriber AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmSession (C++) 202 AMI C MQBACK parameter AmMessage (C++) 206 AmSession (Java) 404 structure 583 AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584	AmSession (C++) 202	AmPolicy (Java) 427	AMI C commit parameter
deleteSession AmReceiver (C++) 217 AmI C handle poison message AmSessionFactory (C++) 198 AmReceiver (Java) 417 parameter structure 579 deleteSubscriber AmSession (C++) 202 AmSender (Java) 413 deleteTopic AmMessage (C++) 206 AmSession (Java) 404 AmMessage (Java) 407 AmSubscriber (C++) 224 AmSubscriber (C++) 224 AmI C MQBEGIN parameter AmI C MQBEGIN parameter structure 583 AmI C MQBEGIN parameter AmI C MQBEGIN parameter AmI C MQCLOSE parameter structure 583 AmI C MQCLOSE parameter structure 583 AmI C MQCLOSE parameter structure 584	deleteSender	AmPublisher (C++) 222	structure 578
AmSessionFactory (C++) 198 AmReceiver (Java) 417 parameter structure 579 deleteSubscriber AmSender (C++) 213 AMI C MQBACK parameter AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmSession (C++) 202 AMI C MQBEGIN parameter AmMessage (C++) 206 AmSession (Java) 404 structure 583 AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584	AmSession (C++) 202	AmPublisher (Java) 422	AMI C element 562
AmSessionFactory (C++) 198 AmReceiver (Java) 417 parameter structure 579 deleteSubscriber AmSender (C++) 213 AMI C MQBACK parameter AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmSession (C++) 202 AMI C MQBEGIN parameter AmMessage (C++) 206 AmSession (Java) 404 structure 583 AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584		AmReceiver (C++) 217	AMI C handle poison message
deleteSubscriber AmSession (C++) 202 AmSender (C++) 213 AMI C MQBACK parameter AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmSession (C++) 202 AMI C MQBEGIN parameter AmMessage (C++) 206 AmSession (Java) 404 structure 583 AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584			<u>.</u>
AmSession (C++) 202 AmSender (Java) 413 structure 583 deleteTopic AmSession (C++) 202 AMI C MQBEGIN parameter AmMessage (C++) 206 AmSession (Java) 404 structure 583 AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584		- ,	-
deleteTopicAmSession (C++)202AMI C MQBEGIN parameterAmMessage (C++)206AmSession (Java)404structure583AmMessage (Java)407AmSubscriber (C++)224AMI C MQCLOSE parameterdeleting C++ objects166AmSubscriber (Java)424structure584			
AmMessage (C++) 206 AmSession (Java) 404 structure 583 AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584	· · ·	- ,	
AmMessage (Java) 407 AmSubscriber (C++) 224 AMI C MQCLOSE parameter deleting C++ objects 166 AmSubscriber (Java) 424 structure 584		, ,	
deleting C++ objects 166 AmSubscriber (Java) 424 structure 584		- ,	
		, ,	<u> -</u>
unectory information tiee, EDAT 312 Encounty attribute 494		- /	Structure 504
	directory information free, LDAI 312	Encounty attribute 474	

AMI C MQCMIT parameter AmStatus (C++) 232 AmSession (Jav structure 584 AmStatus (Java) 431 AmSubscriber	(continued)
structure 584 AmStatus (Java) 431 AmSubscriber	
· · · · · · · · · · · · · · · · · · ·	
AMI C MQCONN parameter AmWarningException (C++) 237 AmSubscriber	(Java) 424
structure 585 AmWarningException (Java) 434 getLocalHost	(java) 121
AMI C MQCONNX parameter getCorrelationId AmSessionFact	tory (C++) 198
structure 585 AmMessage (C++) 206 AmSessionFact	
AMI C MQDISC parameter AmMessage (Java) 407 getMessageId	tory (Java) 400
structure 586 getDataLength AmMessage (C	C++) 208
AMI C MQGET parameter AmMessage (C++) 206 AmMessage (Jacobs Ammess	ava) 409
structure 586 AmMessage (Java) 407 getMethodName	tion (C++) 226
AMI C MQINQ parameter getDataOffset AmErrorExcep	
structure 587 AmMessage (C++) 206 AmErrorExcep	- /
AMI C MQOPEN parameter AmMessage (Java) 407 AmException (
structure 587 getDefinitionType AmException (
	ception (C++) 237
	ception (Java) 434
AMI C MQPUT1 parameter AmSubscriber (C++) 224 getName	T. (G.)
structure 588 AmSubscriber (Java) 424 AmDistribution	1 1
AMI C MQSET parameter getElement AmDistribution	- ,
structure 589 AmMessage (C++) 207 AmElement (C	,
AMI C open connection parameter AmMessage (Java) 407 AmElement (Ja	· ·
structure 579 getElementCCSID AmMessage (C	
AMI C open distribution list AmMessage (C++) 207 AmMessage (James age (James age (C++) 207 AmMessage (James age (James age (C++) 207 Ammessage (James age (ava) 409
parameter structure 579 getElementCount AmObject (C+-	+) 231
AMI C open service parameter AmMessage (C++) 207 AmObject (Jav.	a) 430
structure 580 AmMessage (Java) 407 AmPolicy (C+-	+) 227
AMI C receive from service parameter getEncoding AmPolicy (Java	a) 427
structure 581 AmMessage (C++) 207 AmPublisher (C++) 222
AMI C rollback parameter AmMessage (Java) 408 AmPublisher (Java) 422
structure 581 AmPublisher (C++) 222 AmReceiver (C	C++) 218
AMI C send to distribution list AmPublisher (Java) 422 AmReceiver (Java)	ava) 418
parameter structure 582 AmSender (C++) 213 AmSender (C+	+) 214
AMI C send to service parameter AmSender (Java) 413 AmSender (Java)	va) 414
structure 582 AmSubscriber (C++) 224 AmSession (C+	++) 203
file transfer AmSubscriber (Java) 424 AmSession (Jav	va) 404
C 21 getFactoryName AmSubscriber	
C++ 169 AmSessionFactory (C++) 198 AmSubscriber	(Java) 425
COBOL 251 AmSessionFactory (Java) 400 getNamedElement	
Ambession actory (java) 400 gen amedicinen	
	t
Java 377 getFilter AmMessage (C	t C++) 208
Java 377 getFilter AmMessage (C filters 479 AmMessage (C++) 207 AmMessage (Java 479 AmMessage (C++) 207 AmMessage (Java 479 AmMessage (C++) 207 Am	t C++) 208 ava) 409
Java377getFilterAmMessage (Cfilters479AmMessage (C++)207AmMessage (Jfilters for publish/subscribe479AmMessage (Java)408getNamedElement	t C++) 208 ava) 409 tCount
Java377getFilterAmMessage (Cfilters479AmMessage (C++)207AmMessage (Jfilters for publish/subscribe479AmMessage (Java)408getNamedElementformat constants563getFilterCountAmMessage (C	t C++) 208 ava) 409 tCount C++) 208
Java 377 getFilter AmMessage (Cfilters 479 AmMessage (Cfilters for publish/subscribe 479 AmMessage (Java) 408 getNamedElement format constants 563 getFilterCount AmMessage (CfilterCount Ammessage (C	t C++) 208 ava) 409 tCount C++) 208
Java 377 getFilter AmMessage (Cffilters 479 AmMessage (Cffilters for publish/subscribe 479 AmMessage (Java) 408 getNamedElement format constants 563 getFilterCount AmMessage (CffilterCount AmMessage (CffilterCount AmMessage (CffilterCount AmMessage (Java) 408 getQueueName (Java) 408 getQueueName (Java) 408 getQueueName (Java) 408	t C++) 208 ava) 409 tCount C++) 208 ava) 409
Java 377 getFilter AmMessage (C++) 207 AmMessage (G-+) 207 filters for publish/subscribe 479 format constants 563 getFilterCount AmMessage (Java) 408 getNamedElement AmMessage (C++) 207 AmMessage (C-+) 207 AmMessage (G-+) 207 AmMessage (G-+) 207 AmMessage (G-+) 207 AmReceiver (G-+) 207 AmReceiver (G-+) 207	t C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218
Java 377 getFilter AmMessage (C++) 207 AmMessage (Java) 408 getNamedElement format constants 563 getFilterCount AmMessage (C++) 207 AmMessage (Java) 408 getNamedElement AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (Java) 408 getQueueName getFormat AmReceiver (Java) 408 getCCSID AmMessage (L++) 207 AmReceiver (Java) 408 AmSubscriber	t C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418
Java 377 filters 479 filters for publish/subscribe 479 format constants 563 GetCCSID AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (Java) 408 getNamedElement AmMessage (Java) 408 getFilterCount AmMessage (C++) 207 AmMessage (Java) 408 getQueueName GetCCSID AmMessage (C++) 207 AmMessage (Java) 408 AmSubscriber getGroupStatus	t C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225
Java 377 filters 479 filters for publish/subscribe 479 format constants 563 GetCCSID AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (G++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (G++) 208 getReasonCode	t C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225
Java 377 filters 479 filters 479 filters for publish/subscribe 479 format constants 563 getFilterCount AmMessage (C++) 207 AmMessage (Java) 408 getNamedElement AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (Java) 408 getQueueName getCCSID AmMessage (C++) 207 AmMe	t C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425
Java 377 filters 479 filters for publish/subscribe 479 format constants 563 GetFilter Count AmMessage (C++) 207 AmMessage (Java) 408 getNamedElement GetFilterCount AmMessage (C++) 207 AmMessage (Java) 408 getQueueName getFormat AmMessage (Java) 408 getQueueName getCCSID AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (Java) 408 AmMessage (C++) 207 AmMessage (Java) 408 AmSubscriber AmMessage (Java) 408 AmSubscriber AmPublisher (C++) 222 AmPublisher (Java) 422 AmMessage (Java) 408 AmErrorExcep	t C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425 tion (C++) 236
Java 377 filters 479 filters for publish/subscribe 479 format constants 563 GetFilterCount AmMessage (C++) 207 AmMessage (Java) 408 getNamedElement GetFilterCount AmMessage (C++) 207 AmMessage (Java) 408 getQueueName getFormat AmMessage (Java) 408 getQueueName getCCSID AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (Java) 408 getQueueName getCCSID AmMessage (C++) 207 AmMessage (Java) 408 AmSubscriber AmMessage (Java) 408 AmSubscriber AmMessage (Java) 407 AmMessage (Java) 408 AmMessage (Java) 408 AmSubscriber AmPublisher (C++) 222 AmPublisher (Java) 422 AmSender (C++) 213 getLastErrorStatus AmErrorExcep getLastErrorStatus	t t t t 208 ava) 409 tCount C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425 tion (C++) 236 tion (Java) 433
Java 377 filters 479 filters for publish/subscribe 479 format constants 563 GetFilterCount AmMessage (C++) 207 AmMessage (Java) 408 getNamedElement getFilterCount AmMessage (C++) 207 AmMessage (Java) 408 getQueueName getFormat AmMessage (Java) 408 getQueueName getCCSID AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (Java) 408 getQueueName getGroupStatus AmMessage (Java) 408 AmSubscriber AmMessage (Java) 407 AmMessage (Java) 407 AmPublisher (C++) 222 AmPublisher (Java) 422 AmSender (C++) 213 AmSender (Iava) 413 AmDistributionList (C++) 220 AmException (Java) 413	tt C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425 tion (C++) 236 tion (Java) 433 (C++) 235
Java 377 filters 479 filters 479 filters for publish/subscribe 479 format constants 563 GetFilterCount AmMessage (C++) 207 AmMessage (G++) 207 AmMessage (G++) 207 AmMessage (C++) 207 AmMessage (G++) 207 AmMessage (Java) 408 getQueueName getFormat AmReceiver (G AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (G++) 208 AmMessage (Java) 408 AmSubscriber getGroupStatus AmSubscriber AmMessage (C++) 208 getReasonCode AmMessage (C++) 208 AmMessage (C++) 208 AmErrorExcep getLastErrorStatus AmErrorExcep AmDistributionList (C++) 220 AmException (AmException (AmException (C++) 224 AmMessage (C++) 208 AmStatus (C++) 208 AmStatus (C++) 208 AmStatus (C++) 208 AmStatus (C++) 208	tt C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425 tion (C++) 236 tion (Java) 433 (C++) 235 (Java) 432
Java 377 filters 479 filters for publish/subscribe 479 format constants 563 GetFilterCount AmMessage (C++) 207 AmMessage (Java) 408 getNamedElement getFilterCount AmMessage (C++) 207 AmMessage (G++) 207 AmMessage (Java) 408 getQueueName getFormat AmMessage (Java) 408 getQueueName getFormat AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (Java) 408 getGroupStatus AmSubscriber AmPublisher (C++) 222 AmPublisher (Java) 422 AmSender (C++) 213 AmSender (Java) 413 AmSubscriber (C++) 224 AmSubscriber (Java) 413 AmSubscriber (Lava) 413 AmSubscriber (Java) 424 AmSubscriber (Java) 424 AmSubscriber (Java) 424 AmMessage (C++) 208 AmMessage (C++) 220 AmException (Ammessage (C++) 220 AmException (Ammessage (C++) 208 AmStatus (C++) 208	tt C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425 tion (C++) 236 tion (Java) 433 (C++) 235 (Java) 432 +) 232
Java 377 filters 479 filters 479 filters for publish/subscribe 479 format constants 563 GetFilterCount AmMessage (C++) 207 AmMessage (G-+) 207 AmMessage (G-+) 207 AmMessage (G-+) 207 AmMessage (G-+) 207 AmMessage (Java) 408 getQueueName getFormat AmMessage (G-+) 207 AmMessage (Java) 408 AmMessage (Java) 408 AmMessage (Java) 408 AmSubscriber AmMessage (Java) 407 AmPublisher (C++) 222 AmPublisher (Java) 422 AmSender (C++) 213 AmSender (Java) 413 AmSubscriber (Java) 413 AmSubscriber (C++) 224 AmSubscriber (Java) 413 AmSubscriber (Java) 413 AmSubscriber (Java) 424 AmMessage (Java) 408 AmMessage (C++) 208 AmException (AmMessage (G++) 208 AmException (AmMe	tt C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425 tion (C++) 236 tion (Java) 433 (C++) 235 Ulava) 432 Ulava) 432 Ulava) 432 Ulava) 431
Java 377 filters 479 filters 479 filters for publish/subscribe 479 format constants 563 GetFilterCount AmMessage (C++) 207 AmMessage (G AmMessage (C++) 207 AmMessage (G AmMessage (C++) 207 AmMessage (G AmMessage (Java) 408 getQueueName getFormat AmMessage (Java) 408 getQueueName getFormat AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (G++) 207 AmMessage (Java) 408 AmMessage (Java) 408 AmMessage (Java) 408 AmMessage (Java) 408 AmSubscriber AmPublisher (C++) 222 AmPublisher (Java) 422 AmSender (C++) 213 AmSender (Java) 413 AmSubscriber (Java) 413 AmSubscriber (C++) 224 AmSubscriber (Java) 413 AmSubscriber (C++) 224 AmSubscriber (Java) 424 AmSubscriber (Java) 424 getClassName AmErrorException (C++) 236 AmObject (C++) 231 AmWarningEx	t C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425 tion (C++) 236 tion (Java) 433 (C++) 235 (Java) 432 +) 232 a) 431 ception (C++) 235
Java 377 filters 479 filters 479 filters for publish/subscribe 479 format constants 563 G G getCCSID AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (G++) 208 AmMessage (Java) 408 AmSubscriber AmPublisher (C++) 222 AmPublisher (G++) 222 AmSender (C++) 222 AmSender (C++) 213 AmSender (G++) 213 AmSubscriber (G++) 214 AmSubscriber (G++) 224 AmSubscriber (G++) 226 AmSubscriber	t C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425 tion (C++) 236 tion (Java) 433 (C++) 235 (Java) 432 +) 232 a) 431 ception (C++) 235
Java 377 filters 479 filters 479 filters for publish/subscribe 479 format constants 563 G getFilterCount AmMessage (C++) 207 AmMessage (Java) 408 getQueueName getFormat AmMessage (Java) 408 getQueueName getFormat AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (Java) 408 AmErrorExcepton (C++) 213 AmSender (Java) 413 AmSubscriber (C++) 224 AmSubscriber (Java) 413 AmSubscriber (Java) 424 getClassName AmErrorException (C++) 236 AmMessage (Java) 408 AmMessage (Java) 408 AmStatus (Java) 429 getClassName AmErrorException (Java) 433 AmPolicy (C++) 231 AmWarningEx AmObject (Java) 430 AmWarningEx AmPolicy (C++) 227 getReasonCode2	t C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425 tion (C++) 236 tion (Java) 433 (C++) 235 Java) 432 +) 232 a) 431 ception (C++) 236 ception (Java) 434
GetCCSID AmMessage (C++) 207 AmMessage (G-+) 208 AmSubscriber (G-+) 222 AmMessage (G-+) 208 AmErrorException (G-+) 213 AmSubscriber (G-+) 213 AmSubscriber (G-+) 213 AmSubscriber (G-+) 224 AmMessage (G-+) 208 AmException (G-+) 236 AmMessage (G-+) 231 AmMessage (G-+) 208 AmStatus (G-+) 231 AmMessage (G-+)	t t t t t t t t t t t t t t t t t t t
Java 377 filters 479 filters 479 filters for publish/subscribe 479 format constants 563 GetFilter Count AmMessage (Java) 408 getNamedElement AmMessage (Garana AmSubscriber AmMessage (Java) 407 AmMessage (Java) 408 AmErrorException (C++) 213 AmSubscriber (Java) 413 AmSubscriber (Java) 413 AmSubscriber (Java) 413 AmSubscriber (Java) 413 AmSubscriber (Java) 424 AmMessage (Java) 408 AmErrorException (C++) 224 AmMessage (Java) 408 AmException (Garana AmMessage (Garana AmMessage (Garana AmStatus (Garana AmMessage	t t t t t t t t t t t t t t t t t t t
GetCCSID AmMessage (C++) 207 AmMessage (G++) 207 AmMessage (C++) 207 AmMessage (C++) 207 AmMessage (G++) 208 AmSubscriber (G++) 222 AmMessage (G++) 208 AmMessage (G++) 208 AmMessage (G++) 208 AmErrorException (G++) 213 AmSubscriber (G++) 213 AmSubscriber (G++) 213 AmSubscriber (G++) 213 AmSubscriber (G++) 224 AmMessage (G++) 208 AmException (G++) 244 AmMessage (G++) 208 AmMessage (G++) 208 AmStatus (G++) 208 AmErrorException (G++) 236 AmMerrorException (G++) 236 AmMerrorException (G++) 236 AmPolicy (G++) 227 AmStatus (G++) 227 AmStatus (G++) 227 AmStatus (G++) 227 AmStatus (G++) 228 AmPollisher (G++) 222 AmPollisher (G++) 222 AmPollisher (G++) 222 AmStatus (G++) 222 AmStatus (G++) 223 AmPublisher (G++) 222 AmPublisher (G++) 222 AmStatus (G++) 222 AmStatus (G++) 223 AmPublisher (G++) 222 AmStatus (G++) 223 AmPublisher (G++) 222 AmPublisher (G++) 222 AmStatus (G++) 223 AmPublisher (G++) 222 AmPublisher (G++) 222 AmStatus (G++) 223 AmPublisher (G++) 222 AmStatus (G++) 224 AmStatus (G++) 244 AmPublisher (G++) 222 AmPublisher (G++) 222 AmStatus (G++) 244 AmPublisher (G++) 222 AmPublisher (G++) 222 AmStatus (G++) 244 AmPublisher (G++) 222 AmPublisher (G++) 222 AmPublisher (G++) 222 AmPublisher (G++) 222 AmStatus (G++) 244 AmPublisher (G++) 242 AmStatus (G++) 244 AmPublisher (G++) 242 AmStatus (G++) 244 AmPublisher (G++) 242 AmStatus (G++) 244 AmStatus (G++) 244 AmPublisher (G++)	t C++) 208 ava) 409 tCount C++) 208 ava) 409 C++) 218 ava) 418 (C++) 225 (Java) 425 tion (C++) 236 tion (Java) 433 (C++) 235 Java) 432 +) 232 a) 431 ception (C++) 236 ception (C++) 236 ception (Java) 434 +) 232 a) 431
GetCCSID AmMessage (C++) 207 AmMessage (G++) 208 AmSubscriber AmPublisher (C++) 222 AmPublisher (Java) 422 AmSender (G++) 213 AmSender (Java) 413 AmSubscriber (G++) 213 AmSubscriber (G++) 224 AmSubscriber (G++) 224 AmSubscriber (G++) 224 AmSubscriber (G++) 224 AmSubscriber (Java) 424 AmMessage (G++) 208 AmException (G++) 236 AmMessage (G++) 208 AmException (G++) 236 AmErrorException (G++) 236 AmErrorException (G++) 236 AmErrorException (G++) 235 AmErrorException (G++) 235 AmException (G++) 235 AmException (G++) 237 AmPolicy (Java) 427 AmStatus (G++) 227 AmStatus (G++) 227 AmPublisher (G++) 222 AmStatus (G++) 227 AmPublisher (G++) 227 AmStatus (G++) 227 AmPublisher (G++) 227 AmStatus (G+-) 237 AmPublisher (G++) 222 AmStatus (G+-) 237 AmPublisher (G++) 228 AmMessage (G-+) 218 AmMessage (G-+) 2	t t t t t t t t t t t t t t t t t t t
Java 377 filters 479 AmMessage (C++) 207 AmMessage (G filters for publish/subscribe 479 AmMessage (Java) 408 getFilterCount AmMessage (G AmMessage (C++) 207 AmMessage (G AmMessage (G++) 208 AmMessage (G++) 208 AmMessage (G++) 208 AmPublisher (C++) 222 AmPublisher (G++) 222 AmPublisher (G++) 213 AmSender (C++) 213 AmSender (G++) 213 AmSubscriber (G++) 224 AmSubscriber (Java) 413 AmSubscriber (Java) 413 AmSubscriber (Java) 424 getClassName AmErrorException (C++) 236 AmErrorException (G++) 236 AmErrorException (G++) 235 AmException (G++) 235 AmException (G++) 235 AmException (G++) 235 AmException (G++) 237 AmWarningException (G++) 236 AmReceiver (C++) 218 AmReceiver (G++) 218 AmReceiver (G++) 218 AmReceiver (G++) 218 AmMessage (G AmReceiver (G++) 218 A	t t t t t t t t t t t t t t t t t t t
Java 377 getFilter	t t t t t t t t t t t t t t t t t t t
Java 377 filters 479 AmMessage (C++) 207 AmMessage (G filters for publish/subscribe 479 AmMessage (Java) 408 getFilterCount AmMessage (C++) 207 AmMessage (G AmMessage (C++) 207 AmMessage (G AmMessage (G++) 207 AmMessage (Java) 408 getFormat AmMessage (Java) 408 getFormat AmMessage (G++) 207 AmMessage (G++) 207 AmMessage (G++) 207 AmMessage (G++) 207 AmMessage (G++) 208 AmMessage (Java) 408 AmSubscriber AmMessage (Java) 408 AmSubscriber AmPublisher (C++) 222 AmSender (C++) 213 AmSender (Java) 413 AmSubscriber (Java) 413 AmSubscriber (Java) 413 AmSubscriber (Java) 424 getClassName AmErrorException (C++) 236 AmErrorException (G++) 236 AmErrorException (G++) 235 AmException (G++) 235 AmException (Java) 434 getCompletionCode AmPublisher (Java) 434 getCompletionCode AmErrorException (G++) 236 AmPublisher (Java) 432 AmPublisher (Java) 434 getCompletionCode AmErrorException (C++) 236 AmReceiver (C++) 222 AmPublisher (Java) 434 getCompletionCode AmReceiver (C++) 218 AmMessage (Java) 408 AmStatus (Java) 430 AmWarningException (Java) 433 AmPolicy (G++) 227 AmPublisher (Java) 427 AmStatus (Java) 430 AmWarningException (Java) 434 getCompletionCode AmReceiver (C++) 218 AmMessage (Java) 427 AmStatus (Java) 430 AmPublisher (Java) 430 AmPublisher (Java) 430 AmPublisher (Java) 427 AmStatus (Java) 430 AmPublisher (Java) 421 AmPublisher (Java) 422 AmPublisher (Java) 422 AmPublisher (Java) 423 AmPublisher (Java) 424 AmPublisher (Java) 425 AmPublisher (Java) 426 AmMessage (Java) 427 AmStatus (Java) 430 AmPublisher (Java) 421 AmPublisher (Java) 422 AmPublisher (Java) 421 AmPublisher (Java) 422 AmPublisher (Java) 421 AmPublisher (Java) 421 AmPublisher (Java) 421 AmPublisher (Java) 421 AmPublisher	t t t t t t t t t t t t t t t t t t t

getSender	high-level interface (C) (continued)	installation (continued)
AmDistributionList (C++) 220	amPublish 50	AIX 444
AmDistributionList (Java) 420	amReceiveFile 51	AMI Administration Tool 477
getSenderCount	amReceiveMsg 53	AS/400 448
AmDistributionList (C++) 220	amReceivePublication 55	HP-UX 453
AmDistributionList (Java) 420	amReceiveRequest 57	OS/390 458
getSource	amSendFile 59	policy handler 36
AmErrorException (C++) 236	amSendMsg 60	prerequisites 441
AmErrorException (Java) 433	amSendRequest 61	schema, Active Directory 508
AmException (C++) 235	amSendResponse 62	schema, SecureWay Directory
AmException (Java) 432	amSubscribe 63	server 507
AmWarningException (C++) 237	amTerminate 64	SecureWay Directory client 506
AmWarningException (Java) 434	amUnsubscribe 65	Solaris 461
getTopic	overview 42	Windows 466
AmMessage (C++) 209	reference information 43	integer properties
AmMessage (Java) 409	high-level interface (COBOL)	connection object 592
getTopicCount	AMHBACK 263	message object 595
AmMessage (C++) 209	AMHBEGIN 264	policy object 603
AmMessage (Java) 409	AMHBRMS 265	service object 611
getTraceLevel	AMHCMIT 267	integer property true/false
AmSession (C++) $\frac{203}{404}$	AMHINIT 268	constants 575
AmSession (Java) 404	AMHPB 269	interface
AmSessionFactory (C++) 198	AMHRCFL 270	C++ 181, 197
AmSessionFactory (Java) 400	AMHRCMS 272	C high-level 41, 42
getTraceLocation	AMHRCPO 274	C object 67, 81
AmSession (C++) 203 AmSession (Java) 404	AMHRCRQ 276 AMHSB 282	COBOL high-level 259, 260 COBOL object 285, 299
AmSessionFactory (C++) 198	AMHSNFL 278	Java 385, 399
AmSessionFactory (Java) 400	AMHSNMS 279	interoperability 3
getType	AMHSNRQ 280	Invocation Parameters attribute 503
AmMessage (C++) 209	AMHSNRS 281	invocation point constants 571
AmMessage (Java) 410	AMHTERM 283	invocation points
getValue	AMHUN 284	AMI operation 150
AmElement (C++) 230	equivalent object interface	functions (amPhlXxx) 153
AmElement (Java) 429	functions 296	policy handler 150
getVersion	overview 260	post-transport 151
AmElement (C++) 230	reference information 262	pre-transport 151
AmElement (Java) 429	using 243	I I
getWaitTime	HP-UX	
AmPolicy (C++) 227	C++ applications 177	.J
AmPolicy (Java) 427	C applications 32	· (:1 /I) 202
glossary 629	installation 453	jar file (Java) 382
group status constants 563	Java applications 382	Java applications
	prerequisites 441	AIX 382
	HTML (Hypertext Markup	AS/400 382
Н	Language) 634	HP-UX 382
handle constants E62 E71	Hypertext Markup Language	Solaris 382
handle constants 563, 571 Handle Poison Message attribute 500	(HTML) 634	Windows 382 Java interface
handle property limit constants 571		overview 385
Handler attribute 503	_	reference information 399
header file	1	using 371
C 29	IBM SecureWay Directory client	using 571
C++ 175	installation 506	
helper classes	IBM SecureWay Directory server 506	L
C++ 164, 197	schema deletion 509	_
Java 372, 399	schema installation 507	LDAP. Lightweight Directory Access
helper macros 113	ICONV_UCS2_PREFIX environment	Protocol 505
high-level interface	variable 458	LDAP error codes
equivalent object interface	Implicit Open attribute 498, 500	description 537
functions 78	include file	values 556
using 13	C 29	Idapmodify command 507
high-level interface (C)	C++ 175	Leave Queue Open attribute 498, 500
amBackout 44	Inform If Retained attribute 502	Library functions, policy handler, 151
amBegin 45	initial values for structures 29	library functions, policy handler 151
amBrowseMsg 46	Initialization Parameters attribute 504	Lightweight Directory Access Protocol (LDAP) 505
amCommit 48	installation	installation 506
amInitialize 49	administration tool 477	hounded: 500

Lightweight Directory Access Protocol	message interface (COBOL)	messages, request/response (continued)
(LDAP) (continued)	AMMSADEL 314	Java 376
uninstallation 509	AMMSADFI 314	messages, sending
updating from a repository 509	AMMSADTO 315	C 16
linking a policy handler 36	AMMSCLEC 315	C++ 166
local host cache (OS/390) 475	AMMSDEEL 315	COBOL 246
local host file 471	AMMSDEFI 316	Java 373
local host file (OS/390) 473	AMMSDENE 316	Microsoft Active Directory 506
	AMMSDETO 317	disabling the schema 509
R/I	AMMSGELC 317 AMMSGTCC 317	schema installation 508 model of the AMI 7
M	AMMSGTCI 318	Model Queue Name attribute 494
macros, helper 113	AMMSGTDL 318	MQSeries client
maximum string length constant 576	AMMSGTDO 318	connecting to 478
Message Content Descriptor tag	AMMSGTEC 319	prerequisites 442
names 563	AMMSGTEL 319	MQSeries environment 442
Message Context attribute 498	AMMSGTEN 320	MQSeries function calls
message interface	AMMSGTFC 320	C 27
overview (C) 70	AMMSGTFI 321	C++ 173
overview (C++) 185	AMMSGTFO 321	COBOL 256
overview (COBOL) 288	AMMSGTGS 322	Java 381
overview (Java) 388	AMMSGTLE 322	MQSeries Integrator V2 494
message interface (C)	AMMSGTMI 323	Migrating API applications to 480
amMsgAddElement 95	AMMSGTNA 323	Using the AMI with 478
amMsgAddFilter 95	AMMSGTNC 324	MQSeries Integrator Version 1,
AmMsgAddStreamName 113	AMMSGTNE 324	using 478
amMsgAddTopic 96	AMMSGTRC 325	MQSeries objects, creating 480
amMsgClearErrorCodes 96	AMMSGTTC 326	MQSeries publications 633
amMsgDeleteElement 96	AMMSGTTO 325	MQSeries Publish/Subscribe 478
amMsgDeleteFilter 97 amMsgDeleteNamedElement 97	AMMSGTTY 326	MQSeries server
amMsgDeleteTopic 98	AMMSREBY 327	connecting to 478
amMsgGetCCSID 98	AMMSRS 327	prerequisites 442
amMsgGetCc5iD 98	AMMSSELC 328	multithreading
amMsgGetDataLength 99	AMMSSTCC 328	C 27
amMsgGetDataOffset 99	AMMSSTCI 328	C++ 173
amMsgGetElement 99	AMMSSTDO 329	COBOL 256
amMsgGetElementCCSID 100	AMMSSTEN 329	Java 381
amMsgGetElementCount 100	AMMSSTFO 330	
amMsgGetEncoding 100	AMMSSTGS 330	N.I.
amMsgGetFilter 101	AMMSWRBY 331	N
amMsgGetFilterCount 101	message object	Name attribute
amMsgGetFormat 102	functions 617	distribution list 496
amMsgGetGroupStatus 102	integer properties 595	policy 497
amMsgGetLastError 103	string properties 602	policy handler 504
amMsgGetMsgId 103	message object integer property identifiers 573	publisher 496
amMsgGetName 104	message object integer property value	service point 494
amMsgGetNamedElement 104	constants 572, 575	name/value elements
amMsgGetNamedElementCount 105	message object string property	C 24
AmMsgGetPubTimeStamp 113	identifiers 573	C++ 171
amMsgGetReportCode 105	message types 563	COBOL 253
AmMsgGetStreamName 114	messages 4	Java 378
amMsgGetTopic 106	messages, poison 500	New Correl Id attribute 498
amMsgGetTopicCount 107	messages, publish/subscribe	New Publications Only attribute 502
amMsgGetType 106	C 22	
amMsgReadBytes 107	C++ 170	
amMsgReset 107	COBOL 251	O
amMsgSetCCSID 108	Java 377	OAMAS subset 28
amMsgSetCorrelId 108	messages, receiving	oamasami.h header 28
amMsgSetDataOffset 108	C 18	object interface
amMsgSetElementCCSID 109	C++ 167	overview 67
amMsgSetEncoding 109	COBOL 248	reference information 81
amMsgSetFormat 110	Java 375	object interface (COBOL)
amMsgSetGroupStatus 110 amMsgSetReportCode 111	messages, request/response	overview 285
amMsgSetType 111	C 19	reference information 299
amMsgWriteBytes 112	C++ 168	object-style interface 13
helper macros 113	COBOL 250	object-style interface (COBOL) 243

objects	policy handler	Publish On Request Only attribute 502
C 13	attributes 504	publish/subscribe
C++ 163	compiling, linking and installing 36	constants 563
COBOL 243	defining 491	content-based 479
Java 371	overview 149	filters 479
open AmDistributionList (C++) 220	sample, UNIX and Windows 483 sample program 487	introduction 5 using 478
AmDistributionList (C++) 220 AmDistributionList (Java) 420	policy handler continuation codes 577	publish/subscribe messaging
AmPublisher (C++) 223	policy handler interface (C)	C 22
AmPublisher (Java) 423	AMI extensions 157	C++ 170
AmReceiver (C++) 218	invocation points 150	COBOL 251
AmReceiver (Java) 418	library functions 151	Java 377
AmSender (C++) 214	using 149	Publish To Others Only attribute 503
AmSender (Java) 414	policy handler transport types 577	publisher definition 496
AmSession (C++) 203	policy interface	publisher interface
AmSession (Java) 404	overview (C) 77	overview (C) 75
AmSubscriber (C++) 225	overview (C++) 192	overview (C++) 190
AmSubscriber (Java) 425	overview (COBOL) 295	overview (COBOL) 293
Open Shared attribute 500	overview (Java) 395	overview (Java) 393
opening a session C 16	policy interface (C) amPolClearErrorCodes 147	publisher interface (C) amPubClearErrorCodes 136
C++ 165	amPolGetLastError 147	amPubClose 136
COBOL 246	amPolGetName 148	amPubGetCCSID 136
Java 373	amPolGetWaitTime 148	amPubGetEncoding 137
opening objects	amPolSetWaitTime 148	amPubGetLastError 137
C++ 165	policy interface (COBOL)	amPubGetName 138
Java 373	AMPOCLEC 364	amPubOpen 138
operating systems 441	AMPOGTLE 364	amPubPublish 139
OS/390	AMPOGTNA 365	publisher interface (COBOL)
C applications 33	AMPOGTWT 365	AMPBCL 353
COBOL applications 257	AMPOSTWT 366	AMPBCLEC 353
installation 458	policy object	AMPBGTCC 353
prerequisites 441	functions 621	AMPBGTEN 354
schema installation, SecureWay	integer properties 603	AMPROTNA 355
Directory 507, 508	string properties 610	AMPROR 355
OS/390 subsystems, application advice 437	policy object integer property identifiers 574	AMPBOP 355 AMPBPB 356
overloading	policy object integer property value	ANII DI D. 550
C++ 165	constants 574	
Java 373	policy object string property	Q
overview	identifiers 575	
C++ interface 181	Portable Document Format (PDF) 634	Queue Manager Name attribute 494
C high-level interface 42	positive integer property limit	Queue Name attribute 494
C object interface 67	constants 571	
COBOL high-level interface 260	post-transport invocation points, policy	D
COBOL object interface 285	handler 151	K
Java interface 385	PostScript format 635	readBytes
	pre-transport invocation points, policy	AmMessage (C++) 209
P	handler 151 prerequisites	AmMessage (Java) 410 reason codes
-	compilers 442	constants 564
PDF (Portable Document Format) 634	disk space 441	description 537
Persistence attribute 498	MQSeries environment 442	extended C AMI functions 567
point-to-point 5	OAMAS subset 28	Java 570
pointer definition 563	operating systems 441	receive
poison messages 500	Priority attribute 498	AmReceiver (C++) 218, 219, 419
constants 561	problem determination 515	AmReceiver (Java) 418
defining 491	problems, causes of 533	AmSubscriber (C++) 225
general attributes 498	procedural interface 13	AmSubscriber (Java) 425
handler attributes 503	procedural interface (COBOL) 243	receive options 577
initialization attributes 497	programming languages 4	receiver definition 494
publish attributes 503	publications MOSaries 633	receiver interface
receive attributes 500	MQSeries 633 publish	overview (C) 73 overview (C++) 188
send attributes 498	AmPublisher (C++) 223	overview (COBOL) 291
subscribe attributes 502	AmPublisher (Java) 423	overview (Java) 391
summary 6	Publish Locally attribute 503	,

receiver interface (C)	Retain attribute 503	sendFile (continued)
amRcvBrowse 120	Retry Count attribute 498	AmDistributionList (Java) 421
amRcvBrowseSelect 122	Retry Interval attribute 498	AmSender (C++) 215
amRcvClearErrorCodes 123	RF Header 494	AmSender (Java) 415
amRcvClose 124	rollback	sending files
amRcvGose 124 amRcvGetDefnType 124	AmSession (C++) 203	C 21
amRcvGetLastError 125	AmSession (Java) 404	C++ 169
amRcvGetName 125	runtime environment	COBOL 251
amRcvGetQueueName 126	AIX 445	Java 377
amRcvOpen 126	AS/400 448	sending group messages
amRcvReceive 127	HP-UX 454	C 26
amRcvReceiveFile 129	OS/390 458	C++ 173
amRcvSetQueueName 130	Solaris 462	COBOL 256
receiver interface (COBOL)	Windows 467	Java 381
AMRCBR 338		sending messages
AMRCBRSE 340		C 16
AMRCCL 342	S	C++ 166
AMRCCLEC 341	sample programs	COBOL 246
AMRCGTDT 342	AS/400 481	Java 373
AMRCGTLE 343	OS/390 484	service
AMRCGTNA 343	UNIX 481	constants 561
AMRCGTQN 344	Windows 481	defining 491
AMRCOP 344	schema installation	summary 4
AMRCRC 345		service object
AMRCRCFL 346	Active Directory 508	functions 623
AMRCSTQN 347	SecureWay Directory server 507	integer properties 611
Receiver Service attribute 496	SecureWay Directory client	string properties 613
receiving files	installation 506	service object integer property
C 21	SecureWay Directory server 506	identifiers 576
C++ 169	schema deletion 509	service object string property
COBOL 251	schema installation 507	identifiers 576
Java 377	security	service point 494
receiving messages	LDAP 513	Service Type attribute 494
C 18	LDAP, environment variables 513	session factory
C++ 167	Segmentation attribute 498	overview (C++) 182
COBOL 248	send	overview (Java) 386
Java 375	AmDistributionList (C++) 221	session interface
reference information	AmDistributionList (Java) 421	overview (C) 68
C++ interface 197	AmSender (C++) 214	overview (C++) 183
C high-level interface 43	AmSender (Java) 414	overview (COBOL) 286
C object interface 81	sender definition 494	overview (GODOL) 250 overview (Java) 387
,	sender interface	
COBOL object interface 262	overview (C) 72	session interface (C)
COBOL object interface 299	overview (C++) 187	amSesBegin 82
Java interface 399	overview (COBOL) 290	amSesClearErrorCodes 82
Report Data attribute 498	overview (Java) 390	amSesClose 83
Report Type COA attribute 498	sender interface (C)	amSesCommit 83
Report Type COD attribute 498	amSndClearErrorCodes 115	amSesCreate 83
Report Type Exception attribute 498	amSndClose 115	amSesCreateDistList 84
Report Type Expiry attribute 498	amSndGetCCSID 116	amSesCreateMessage 84
repository, using	amSndGetEncoding 116	amSesCreatePolicy 84
C 14	amSndGetLastError 116	amSesCreatePublisher 85
C++ 164	amSndGetName 117	amSesCreateReceiver 85
COBOL 244	amSndOpen 117	amSesCreateSender 86
Java 372	amSndSend 118	amSesCreateSubscriber 86
repository cache (OS/390) 475	amSndSendFile 119	amSesDelete 87
repository file 471	sender interface (COBOL)	amSesDeleteDistList 87
updating LDAP directories 509	AMSNCL 333	amSesDeleteMessage 87
repository file (OS/390) 473	AMSNCLEC 332	amSesDeletePolicy 88
request/response messaging	AMSNGTCC 333	amSesDeletePublisher 88
C 19	AMSNGTEN 333	amSesDeleteReceiver 88
C++ 168	AMSNGTLE 334	amSesDeleteSender 89
COBOL 250	AMSNGTNA 334	amSesDeleteSubscriber 89
Java 376	AMSNOP 335	amSesGetDistListHandle 89
reset	AMSNSN 335	amSesGetLastError 90
AmMessage (C++) 209	AMSNSNFL 336	amSesGetMessageHandle 90
AmMessage (Java) 410	sendFile	amSesGetPolicyHandle 91
Response Correl Id attribute 498	AmDistributionList (C++) 221	amSesGetPublisherHandle 91
	1.11.12.13.11.12.11.11.11.11.11.11.11.11.11.11.11.	

session interface (C) (continued) amSesGetReceiverHandle 91	setQueueName (continued) AmSubscriber (C++) 225	structure (continued) AMI C MQOPEN parameter 587
amSesGetSenderHandle 92	AmSubscriber (Java) 425	AMI C MQPUT parameter 588
amSesGetSubscriberHandle 92	setReportCode	AMI C MQPUT1 parameter 588
amSesOpen 92	AmMessage (C++) 211	AMI C MQSET parameter 589
amSesRollback 93	AmMessage (Java) 411	AMI C open connection
transaction coordination 82	setRepository	parameter 579
unit of work 82	AmSessionFactory (C++) 199	AMI C open distribution list
session interface (COBOL)	AmSessionFactory (Java) 401	parameter 579
AMSEBG 300	setTraceLevel	AMI C open service parameter 580
AMSECL 301	AmSessionFactory (C++) 199	AMI C receive from service
AMSECLEC 300	AmSessionFactory (Java) 401	parameter 581
AMSECM 301	setTraceLocation	AMI C rollback parameter 581
AMSECR 302	AmSessionFactory (C++) 199	AMI C send to distribution list
AMSECRDL 302	AmSessionFactory (Java) 401	parameter 582
AMSECRMS 303	setType	AMI C send to service
AMSECRPB 304	AmMessage (C++) 212	parameter 582
AMSECRPO 303	AmMessage (Java) 412	AMI parameter union 580
AMSECRRC 304	setVersion	structure of the AMI
AMSECRSB 305	AmElement (C++) 230	C 13
AMSECRSN 305	AmElement (Java) 429	C++ 163
AMSEDL 306	setWaitTime	COBOL 243
AMSEDLDL 306	AmPolicy (C++) 227	Java 371
AMSEDLMS 306	AmPolicy (Java) 427	structure of this book xxi
AMSEDLPB 307	signed integer property limit	structures, initial values 29
AMSEDLPO 307	constants 572	subscribe
AMSEDLRC 307	simulated group messages 26, 256	AmSubscriber (C++) 226
AMSEDLSB 308	Simulated Group Support attribute 494	AmSubscriber (Java) 426
AMSEDLSN 308	softcopy books 634	content-based 479
AMSEGHDL 308	Solaris	filters 479
AMSEGHMS 309	C++ applications 178	Subscribe Locally attribute 502
AMSEGHPB 310	C applications 34	subscriber definition 496
AMSEGHPO 310	installation 461	subscriber interface
AMSEGHRC 310	Java applications 382	overview (C) 76
AMSEGHSB 311	prerequisites 441	overview (C++) 191
AMSEGHSN 311	schema installation, SecureWay	overview (COBOL) 294
AMSEGTLE 309	Directory 507	overview (Java) 394
AMSEOP 312	Split File attribute 498	subscriber interface (C)
AMSERB 312 transaction coordination 300	string length constants 563 string properties	amSubClearErrorCodes 140 amSubClose 140
unit of work 300	connection object 593	amSubGetCCSID 140
setCCSID	message object 602	amSubGetDefnType 141
AmMessage (C++) 210	policy object 610	amSubGetEncoding 141
AmMessage (Java) 410	service object 613	amSubGetLastError 142
setCorrelationId	structure 580	amSubGetName 142
AmMessage (C++) 210	AMI C 580	amSubGetQueueName 143
AmMessage (Java) 410	AMI C begin parameter 577	amSubOpen 143
setDataOffset	AMI C close connection	amSubReceive 144
AmMessage (C++) 210	parameter 577	amSubSetQueueName 144
AmMessage (Java) 410	AMI C close distribution list	amSubSubscribe 145
setElementCCSID	parameter 578	amSubUnsubscribe 146
AmMessage (C++) 210	AMI C close service parameter 578	subscriber interface (COBOL)
setEncoding	AMI C commit parameter 578	AMSBCL 357
AmMessage (C++) 210	AMI C element 562	AMSBCLEC 357
AmMessage (Java) 411	AMI C handle poison message	AMSBGTCC 358
setFormat	parameter 579	AMSBGTDT 358
AmMessage (C++) 211	AMI C MQBACK parameter 583	AMSBGTEN 359
AmMessage (Java) 411	AMI C MQBEGIN parameter	AMSBGTLE 359
setGroupStatus	structure 583	AMSBGTNA 360
AmMessage (C++) 211	AMI C MQCLOSE parameter 584	AMSBGTQN 360
AmMessage (Java) 411	AMI C MQCMIT parameter 584	AMSBOP 361
setLocalHost	AMI C MQCONN parameter 585	AMSBRC 361
AmSessionFactory (C++) 199	AMI C MQCONNX parameter 585	AMSBSB 362
AmSessionFactory (Java) 400	AMI C MQDISC parameter 586	AMSBSTQN 362
setQueueName	AMI C MQGET parameter	AMSBUN 363
AmReceiver (C++) 219	structure 586	SupportPac 635
AmReceiver (Java) 419	AMI C MQINQ parameter 587	Suppress Registration attribute 503

Syncpoint attribute 498 system default handle synonyms 570 system default names 570 system default objects C 14 C++ 164 COBOL 244 Java 372	Unsubscribe All attribute 502 Use Correl Id As Id attribute 502, 503 using the AMI C 13 C++ 163 C policy handler 149 COBOL 243 Java 371
Т	W
terminology used in this book 629 tool, administration 477 topics, publish/subscribe C 22 C++ 170 COBOL 251 Java 377 toString AmElement (C++) 230 AmElement (Java) 429 AmErrorException (C++) 236 AmErrorException (Java) 433 AmException (C++) 235 AmException (Java) 432 AmStatus (C++) 232 AmStatus (Java) 431 AmWarningException (Java) 434 trace AS/400 516 C++ and Java 518 example 519 UNIX 516 using, OS/390 529 using, UNIX and Windows 515 Windows 517 trace control constants 576 transaction coordination C 82 C++ 200 COBOL 300 Java 402 transaction support C 26 C++ 172 COBOL 256 Java 380 transport invocation points, policy handler 151 Trusted Option attribute 497	Wait For Whole Group attribute 500 Wait Interval Read Only attribute 500 wait time constants 570 what you need to know xxi who this book is for xxi Windows C++ applications 179 C applications 35 installation 466 Java applications 382 prerequisites 441 sample programs 481 schema installation, SecureWay Directory 507 Windows Help 635 writeBytes AmMessage (C++) 212 AmMessage (Java) 412 writing applications C 16 C++ 165 COBOL 246 Java 373 writing applications for OS/390 subsystems 437 writing IMS applications 437
Unicode character conversion 458	
uninstallation, LDAP 509 unit of work C 26, 82 C++ 172, 200 COBOL 256, 300 Java 380, 402 UNIX	
sample programs 481 unsubscribe AmSubscriber (C++) 226 AmSubscriber (Java) 426	

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

• By mail, to this address:

User Technologies Department (MP095) IBM United Kingdom Laboratories Hursley Park WINCHESTER, Hampshire SO21 2JN United Kingdom

- By fax:
 - From outside the U.K., after your international access code use 44–1962–816151
 - From within the U.K., use 01962-816151
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
 - IBMLink[™]: HURSLEY(IDRCF)
 - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- · The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

IBM.®



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

SC34-5604-07

