

MQSeries®



Application Messaging Interface

Note!

Before using this information and the product it supports, be sure to read the general information under “Appendix C. Notices” on page 517.

Sixth edition (December 2000)

This edition applies to IBM® MQSeries Application Messaging Interface Version 1.1, and to any subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1999, 2000. All rights reserved.

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

Contents

Figures xv

Tables xvii

About this book xix

Who this book is for xix

What you need to know to understand this book xix

Structure of this book xix

Appearance of text in this book xx

Summary of changes xxi

Changes for this edition (SC34-5604-05) xxi

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

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

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

Part 1. Introduction 1

Chapter 1. Introduction 3

Main components of the AMI 3

 Sending and receiving messages 3

 Interoperability 3

 Programming languages 4

Description of the AMI 4

 Messages 4

 Services 4

 Policies 6

Application Messaging Interface model 7

Further information 8

Part 2. The C interface 9

Chapter 2. Using the Application

Messaging Interface in C 13

Structure of the AMI 13

 Using the repository 14

 System default objects 14

Writing applications in C 16

 Opening and closing a session 16

 Sending messages 16

 Receiving messages 18

 Request/response messaging 19

 File transfer 21

 Publish/subscribe messaging 22

 Using name/value elements 24

 Error handling 25

 Transaction support 26

 Sending group messages 26

 Other considerations 27

 Using the AMI OAMAS subset 28

Building C applications 29

 AMI include file 29

 Data types 29

 Initial values for structures 29

 C applications on AIX 30

 C applications on AS/400 31

 C applications on HP-UX 32

 C applications on OS/390 33

 C applications on Solaris 34

 C applications on Windows 35

Chapter 3. The C high-level interface 37

Overview of the C high-level interface 38

 Initialize and terminate 38

 Sending messages 38

 Receiving messages 38

 File transfer 38

 Publish/subscribe 38

 Transaction support 38

Reference information for the C high-level interface 39

amBackout 40

amBegin 41

amBrowseMsg 42

amCommit 44

amInitialize 45

amPublish 46

amReceiveFile 47

amReceiveMsg 49

amReceivePublication 51

amReceiveRequest 53

amSendFile 55

amSendMsg 56

amSendRequest 57

amSendResponse 58

amSubscribe 59

amTerminate 60

amUnsubscribe 61

Chapter 4. C object interface overview 63

Session interface functions 64

 Session management 64

 Create objects 64

 Get object handles 64

 Delete objects 65

 Transactional processing 65

 Error handling 65

Message interface functions 66

 Get values 66

 Set values 66

 Reset values 66

 Read and write data 66

 Publish/subscribe topics 67

 Publish/subscribe filters 67

 Publish/subscribe name/value elements 67

 Error handling 67

 Publish/subscribe helper macros 67

Sender interface functions 68

 Open and close 68

Send	68	amSesOpen	88
Get values	68	amSesRollback	89
Error handling	68	Message interface functions	90
Receiver interface functions	69	amMsgAddElement	91
Open and close	69	amMsgAddFilter	91
Receive and browse	69	amMsgAddTopic	92
Get values	69	amMsgClearErrorCodes	92
Set values	69	amMsgDeleteElement	92
Error handling	69	amMsgDeleteFilter	93
Distribution list interface functions	70	amMsgDeleteNamedElement	93
Open and close	70	amMsgDeleteTopic	94
Send	70	amMsgGetCCSID	94
Get values	70	amMsgGetCorrelId	94
Error handling	70	amMsgGetDataLength	95
Publisher interface functions	71	amMsgGetDataOffset	95
Open and close	71	amMsgGetElement	95
Publish	71	amMsgGetElementCCSID	96
Get values	71	amMsgGetElementCount	96
Error handling	71	amMsgGetEncoding	96
Subscriber interface functions	72	amMsgGetFilter	97
Open and close	72	amMsgGetFilterCount	97
Broker messages	72	amMsgGetFormat	98
Get values	72	amMsgGetGroupStatus	98
Set value	72	amMsgGetLastError	99
Error handling	72	amMsgGetMsgId	99
Policy interface functions	73	amMsgGetName	100
Get values	73	amMsgGetNamedElement	100
Set value	73	amMsgGetNamedElementCount	101
Error handling	73	amMsgGetReportCode	101
High-level functions	74	amMsgGetTopic	102
		amMsgGetType	102
Chapter 5. C object interface reference 77		amMsgGetTopicCount	103
Session interface functions	78	amMsgReadBytes	103
amSesBegin	78	amMsgReset	103
amSesClearErrorCodes	78	amMsgSetCCSID	104
amSesClose	79	amMsgSetCorrelId	104
amSesCommit	79	amMsgSetDataOffset	104
amSesCreate	79	amMsgSetElementCCSID	105
amSesCreateDistList	80	amMsgSetEncoding	105
amSesCreateMessage	80	amMsgSetFormat	106
amSesCreatePolicy	80	amMsgSetGroupStatus	106
amSesCreatePublisher	81	amMsgWriteBytes	107
amSesCreateReceiver	81	Message interface helper macros	108
amSesCreateSender	82	AmMsgAddStreamName	108
amSesCreateSubscriber	82	AmMsgGetPubTimeStamp	108
amSesDelete	83	AmMsgGetStreamName	109
amSesDeleteDistList	83	Sender interface functions	110
amSesDeleteMessage	83	amSndClearErrorCodes	110
amSesDeletePolicy	84	amSndClose	110
amSesDeletePublisher	84	amSndGetCCSID	111
amSesDeleteReceiver	84	amSndGetEncoding	111
amSesDeleteSender	85	amSndGetLastError	111
amSesDeleteSubscriber	85	amSndGetName	112
amSesGetDistListHandle	85	amSndOpen	112
amSesGetLastError	86	amSndSend	113
amSesGetMessageHandle	86	amSndSendFile	114
amSesGetPolicyHandle	87	Receiver interface functions	115
amSesGetPublisherHandle	87	amRcvBrowse	115
amSesGetReceiverHandle	87	amRcvBrowseSelect	117
amSesGetSenderHandle	88	amRcvClearErrorCodes	118
amSesGetSubscriberHandle	88	amRcvClose	119

amRcvGetDefnType	119
amRcvGetLastError	120
amRcvGetName	120
amRcvGetQueueName	121
amRcvOpen	121
amRcvReceive	122
amRcvReceiveFile	124
amRcvSetQueueName	125
Distribution list interface functions	126
amDstClearErrorCodes	126
amDstClose	126
amDstGetLastError	127
amDstGetName	127
amDstGetSenderCount	128
amDstGetSenderHandle	128
amDstOpen	128
amDstSend	129
amDstSendFile	130
Publisher interface functions	131
amPubClearErrorCodes	131
amPubClose	131
amPubGetCCSID	131
amPubGetEncoding	132
amPubGetLastError	132
amPubGetName	133
amPubOpen	133
amPubPublish	134
Subscriber interface functions	135
amSubClearErrorCodes	135
amSubClose	135
amSubGetCCSID	135
amSubGetDefnType	136
amSubGetEncoding	136
amSubGetLastError	137
amSubGetName	137
amSubGetQueueName	138
amSubOpen	138
amSubReceive	139
amSubSetQueueName	139
amSubSubscribe	140
amSubUnsubscribe	141
Policy interface functions	142
amPolClearErrorCodes	142
amPolGetLastError	142
amPolGetName	143
amPolGetWaitTime	143
amPolSetWaitTime	143

Part 3. The C++ interface 145

Chapter 6. Using the Application Messaging Interface in C++ 149

Structure of the AMI	149
Base classes	149
Interface and helper classes	150
Exception classes	150
Using the repository	150
System default objects	150
Writing applications in C++	151
Creating and opening objects	151

Deleting objects	152
Sending messages	152
Receiving messages	153
Request/response messaging	154
File transfer	155
Publish/subscribe messaging	156
Using AmElement objects	157
Error handling	157
Transaction support	158
Sending group messages	159
Other considerations	159
Building C++ applications	161
AMI include files	161
C++ applications on AIX	161
C++ applications on AS/400	162
C++ applications on HP-UX	163
C++ applications on Solaris	164
C++ applications on Windows	165

Chapter 7. C++ interface overview . . . 167

Base classes	167
Helper classes	167
Exception classes	167
AmSessionFactory	168
Constructor	168
Session factory management	168
Create and delete session	168
AmSession	169
Session management	169
Create objects	169
Delete objects	169
Transactional processing	169
Error handling	170
AmMessage	171
Get values	171
Set values	171
Reset values	171
Read and write data	171
Publish/subscribe topics	172
Publish/subscribe filters	172
Publish/subscribe name/value elements	172
Error handling	172
AmSender	173
Open and close	173
Send	173
Send file	173
Get values	173
Error handling	173
AmReceiver	174
Open and close	174
Receive and browse	174
Receive file	174
Get values	174
Set value	174
Error handling	174
AmDistributionList	175
Open and close	175
Send	175
Send file	175
Get values	175
Error handling	175

AmPublisher	176
Open and close.	176
Publish	176
Get values	176
Error handling	176
AmSubscriber	177
Open and close.	177
Broker messages	177
Get values	177
Set value	177
Error handling	177
AmPolicy.	178
Policy management	178
Error handling	178
Helper classes	179
AmBytes	179
AmElement	179
AmObject	179
AmStatus.	179
AmString.	180
Exception classes	181
AmException	181
AmErrorException.	181
AmWarningException	181

Chapter 8. C++ interface reference 183

Base classes	183
Helper classes	183
Exception classes	183
AmSessionFactory	184
AmSessionFactory	184
createSession	184
deleteSession	184
getFactoryName	184
getLocalHost	184
getRepository	184
getTraceLevel	184
getTraceLocation	184
setLocalHost.	185
setRepository	185
setTraceLevel	185
setTraceLocation	185
AmSession	186
begin	186
clearErrorCodes	186
close	186
commit	186
createDistributionList.	186
createMessage	187
createPolicy	187
createPublisher	187
createReceiver	187
createSender.	187
createSubscriber	187
deleteDistributionList.	188
deleteMessage	188
deletePolicy	188
deletePublisher	188
deleteReceiver	188
deleteSender.	188
deleteSubscriber	188

enableWarnings	188
getLastErrorStatus	189
getName	189
getTraceLevel	189
getTraceLocation	189
open	189
rollback	189
AmMessage	190
addElement	191
addFilter	191
addTopic	191
clearErrorCodes	191
deleteElement	191
deleteFilter	191
deleteNamedElement.	192
deleteTopic	192
enableWarnings	192
getCCSID.	192
getCorrelationId	192
getDataLength	192
getDataOffset	192
getElement	193
getElementCCSID	193
getElementCount	193
getEncoding	193
getFilter	193
getFilterCount	193
getFormat	193
getGroupStatus.	194
getLastErrorStatus.	194
getMessageId	194
getName	194
getNamedElement.	194
getNamedElementCount.	194
getReportCode	195
getTopic	195
getTopicCount	195
getType	195
readBytes.	195
reset	195
setCCSID.	196
setCorrelationId	196
setDataOffset	196
setElementCCSID	196
setEncoding	196
setFormat	197
setGroupStatus	197
writeBytes	197
AmSender	198
clearErrorCodes	198
close	198
enableWarnings	198
getCCSID.	198
getEncoding	198
getLastErrorStatus.	199
getName	199
open	199
send	199
sendFile	200
AmReceiver	201
browse	201

clearErrorCodes	202	operators	214
close	202	pad	214
enableWarnings	202	AmElement	215
getDefinitionType	202	AmElement	215
getLastErrorStatus	203	getName	215
getName	203	getValue	215
getQueueName	203	getVersion	215
open	203	setVersion	215
receive	203	toString	215
receiveFile	204	AmObject	216
setQueueName	204	clearErrorCodes	216
AmDistributionList	205	getLastErrorStatus	216
clearErrorCodes	205	getName	216
close	205	AmStatus	217
enableWarnings	205	AmStatus	217
getLastErrorStatus	205	getCompletionCode	217
getName	205	getReasonCode	217
getSender	205	getReasonCode2	217
getSenderCount	205	toString	217
open	205	AmString	218
send	206	cat	218
sendFile	206	cmp	218
AmPublisher	207	constructors	218
clearErrorCodes	207	contains	218
close	207	cpy	218
enableWarnings	207	destructor	218
getCCSID	207	operators	219
getEncoding	207	pad	219
getLastErrorStatus	207	split	219
getName	207	strip	219
open	208	length	219
publish	208	text	219
AmSubscriber	209	truncate	219
clearErrorCodes	209	AmException	220
close	209	getClassName	220
enableWarnings	209	getCompletionCode	220
getCCSID	209	getMethodName	220
getDefinitionType	209	getReasonCode	220
getEncoding	209	getSource	220
getLastErrorStatus	209	toString	220
getName	210	AmErrorException	221
getQueueName	210	getClassName	221
open	210	getCompletionCode	221
receive	210	getMethodName	221
setQueueName	210	getReasonCode	221
subscribe	211	getSource	221
unsubscribe	211	toString	221
AmPolicy	212	AmWarningException	222
clearErrorCodes	212	getClassName	222
enableWarnings	212	getCompletionCode	222
getLastErrorStatus	212	getMethodName	222
getName	212	getReasonCode	222
getWaitTime	212	getSource	222
setWaitTime	212	toString	222
AmBytes	213		
cmp	213		
constructors	213		
cpy	214		
dataPtr	214		
destructor	214		
length	214		
		Part 4. The COBOL interface	223
		Chapter 9. Using the Application	
		Messaging Interface in COBOL.	227
		Structure of the AMI	227

AMSEGTLE (get last error codes)	293	AMRCBRSE (browse selection message)	324
AMSEGHMS (get message handle)	293	AMRCCLEC (clear error codes)	325
AMSEGHPO (get policy handle)	294	AMRCCL (close)	326
AMSEGHPB (get publisher handle)	294	AMRCGTD (get definition type)	326
AMSEGHRC (get receiver handle)	294	AMRCGTLE (get last error)	327
AMSEGHSN (get sender handle)	295	AMRCGTNA (get name)	327
AMSEGHSB (get subscriber handle)	295	AMRCGTQN (get queue name)	328
AMSEOP (open)	296	AMRCOP (open)	328
AMSERB (rollback)	296	AMRCRC (receive)	329
Message interface functions	297	AMRCRCFL (receive file)	330
AMMSADEL (add element)	298	AMRCSTQN (set queue name)	331
AMMSADFI (add filter)	298	Distribution list interface functions	332
AMMSADTO (add topic)	299	AMDLCLEC (clear error codes)	332
AMMSCLEC (clear error codes)	299	AMDLCL (close)	332
AMMSDEEL (delete element)	299	AMDLGTLE (get last error)	332
AMMSDEFI (delete filter)	300	AMDLGTNA (get name)	333
AMMSDENE (delete named element)	300	AMDLGTSC (get sender count)	333
AMMSDETO (delete topic)	301	AMDLGTSH (get sender handle)	334
AMMSGELC (get element CCSID)	301	AMDLOP (open)	334
AMMSGTCC (get CCSID)	301	AMDLSN (send)	335
AMMSGTCI (get correl ID)	302	AMDLSNFL (send file)	335
AMMSGTDL (get data length)	302	Usage notes	336
AMMSGTDO (get data offset)	302	Publisher interface functions	337
AMMSGTEL (get element)	303	AMPBCLEC (clear error codes)	337
AMMSGTEC (get element count)	303	AMPBCL (close)	337
AMMSGTEN (get encoding)	304	AMPBGTC (get CCSID)	337
AMMSGTFC (get filter count)	304	AMPBGTEN (get encoding)	338
AMMSGTFI (get filter)	305	AMPBGTLE (get last error)	338
AMMSGTFO (get format)	305	AMPBGTNA (get name)	339
AMMSGTGS (get group status)	306	AMPBOP (open)	339
AMMSGTLE (get last error)	306	AMPBPB (publish)	340
AMMSGTMI (get message ID)	307	Subscriber interface functions	341
AMMSGTNA (get name)	307	AMSBCLEC (clear error codes)	341
AMMSGTNE (get named element)	308	AMSBCL (close)	341
AMMSGTNC (get named element count)	308	AMSBGTCC (get CCSID)	342
AMMSGTRC (get report code)	309	AMSBGTD (get definition type)	342
AMMSGTTO (get topic)	309	AMSBGTEN (get encoding)	343
AMMSGTTC (get topic count)	310	AMSBGTLE (get last error)	343
AMMSGTTY (get type)	310	AMSBGTNA (get name)	344
AMMSREBY (read bytes)	311	AMSBGTQN (get queue name)	344
AMMSRS (reset)	311	AMSBOP (open)	345
AMMSSTCC (set CCSID)	312	AMSBRC (receive)	345
AMMSSTCI (set correl ID)	312	AMSBSTQN (set queue name)	346
AMMSSTDO (set data offset)	312	AMSBSB (subscribe)	346
AMMSSELC (set element ccsid)	313	AMSBUN (unsubscribe)	347
AMMSSTEN (set encoding)	313	Policy interface functions	348
AMMSSTFO (set format)	314	AMPOCLEC (clear error codes)	348
AMMSSTGS (set group status)	314	AMPOGTLE (get last error)	348
AMMSWRBY (write bytes)	315	AMPOGTNA (get name)	349
Sender interface functions	316	AMPOGTWT (get wait time)	349
AMSNCLEC (clear error codes)	316	AMPOSTWT (set wait time)	350
AMSNCL (close)	317		
AMSNGTCC (get CCSID)	317		
AMSNGTEN (get encoding)	317		
AMSNGTLE (get last error)	318		
AMSNGTNA (get name)	318		
AMSNOP (open)	319		
AMSNSN (send)	319		
AMSNSNFL (send file)	320		
Receiver interface functions	322		
AMRCBR (browse)	322		

Part 5. The Java interface 351

Chapter 13. Using the Application

Messaging Interface in Java 355

Structure of the AMI	355
Base classes	355
Interface and helper classes	356
Exception classes	356

Using the repository	356
System default objects	356
Writing applications in Java	357
Creating and opening objects	357
Sending messages	357
Receiving messages	359
Request/response messaging	360
File transfer	361
Publish/subscribe messaging	361
Using AmElement objects	362
Error handling	363
Transaction support	364
Sending group messages	365
Other considerations	365
Building Java applications	366
AMI package for Java	366
Running Java programs	366

Chapter 14. Java interface overview 369

Base classes	369
Helper classes	369
Exception classes	369
AmSessionFactory	370
Constructor	370
Session factory management	370
Create session	370
AmSession	371
Session management	371
Create objects	371
Transactional processing	371
Error handling	371
AmMessage	372
Get values	372
Set values	372
Reset values	372
Read and write data	372
Publish/subscribe filters	372
Publish/subscribe topics	373
Publish/subscribe name/value elements	373
Error handling	373
AmSender	374
Open and close	374
Send	374
Send file	374
Get values	374
Error handling	374
AmReceiver	375
Open and close	375
Receive and browse	375
Receive file	375
Get values	375
Set value	375
Error handling	375
AmDistributionList	376
Open and close	376
Send	376
Send file	376
Get values	376
Error handling	376
AmPublisher	377
Open and close	377

Publish	377
Get values	377
Error handling	377
AmSubscriber	378
Open and close	378
Broker messages	378
Get values	378
Set value	378
Error handling	378
AmPolicy	379
Policy management	379
Error handling	379
Helper classes	380
AmConstants	380
AmElement	380
AmObject	380
AmStatus	380
Exception classes	381
AmException	381
AmErrorException	381
AmWarningException	381

Chapter 15. Java interface reference 383

Base classes	383
Helper classes	383
Exception classes	383
AmSessionFactory	384
AmSessionFactory	384
createSession	384
getFactoryName	384
getLocalHost	384
getRepository	384
getTraceLevel	384
getTraceLocation	384
setLocalHost	384
setRepository	385
setTraceLevel	385
setTraceLocation	385
AmSession	386
begin	386
clearErrorCodes	386
close	386
commit	386
createDistributionList	386
createMessage	387
createPolicy	387
createPublisher	387
createReceiver	387
createSender	387
createSubscriber	387
enableWarnings	388
getLastErrorStatus	388
getName	388
getTraceLevel	388
getTraceLocation	388
open	388
rollback	388
AmMessage	389
addElement	389
addFilter	390
addTopic	390

clearErrorCodes	390	close	403
deleteElement	390	enableWarnings	403
deleteFilter	390	getLastErrorStatus	403
deleteNamedElement	390	getName	403
deleteTopic	391	getSender	403
enableWarnings	391	getSenderCount	403
getCCSID	391	open	403
getCorrelationId	391	send	404
getDataLength	391	sendFile	404
getDataOffset	391	AmPublisher	405
getElement	391	clearErrorCodes	405
getElementCount	391	close	405
getEncoding	392	enableWarnings	405
getFilter	392	getCCSID	405
getFilterCount	392	getEncoding	405
getFormat	392	getLastErrorStatus	405
getGroupStatus	392	getName	405
getLastErrorStatus	392	open	406
getMessageId	393	publish	406
getName	393	AmSubscriber	407
getNamedElement	393	clearErrorCodes	407
getNamedElementCount	393	close	407
getReportCode	393	enableWarnings	407
getTopic	393	getCCSID	407
getTopicCount	393	getDefinitionType	407
getType	394	getEncoding	407
readBytes	394	getLastErrorStatus	407
reset	394	getName	408
setCCSID	394	getQueueName	408
setCorrelationId	394	open	408
setDataOffset	394	receive	408
setEncoding	395	setQueueName	408
setFormat	395	subscribe	409
setGroupStatus	395	unsubscribe	409
writeBytes	395	AmPolicy	410
AmSender	396	clearErrorCodes	410
clearErrorCodes	396	enableWarnings	410
close	396	getLastErrorStatus	410
enableWarnings	396	getName	410
getCCSID	396	getWaitTime	410
getEncoding	396	setWaitTime	410
getLastErrorStatus	397	AmConstants	411
getName	397	AmElement	412
open	397	AmElement	412
send	397	getName	412
sendFile	398	getValue	412
AmReceiver	399	getVersion	412
browse	399	setVersion	412
clearErrorCodes	400	toString	412
close	400	AmObject	413
enableWarnings	400	clearErrorCodes	413
getDefinitionType	400	getLastErrorStatus	413
getLastErrorStatus	401	getName	413
getName	401	AmStatus	414
getQueueName	401	AmStatus	414
open	401	getCompletionCode	414
receive	401	getReasonCode	414
receiveFile	402	getReasonCode2	414
setQueueName	402	toString	414
AmDistributionList	403	AmException	415
clearErrorCodes	403	getClassName	415

getCompletionCode	415
getMethodName	415
getReasonCode	415
getSource	415
toString	415
AmErrorException	416
getClassName	416
getCompletionCode	416
getMethodName	416
getReasonCode	416
getSource	416
toString	416
AmWarningException	417
getClassName	417
getCompletionCode	417
getMethodName	417
getReasonCode	417
getSource	417
toString	417

Part 6. OS/390 Subsystems 419

Chapter 16. Writing applications for OS/390 subsystems 421

Writing IMS applications using AMI	421
Writing CICS applications using AMI	421
Writing batch applications using AMI	422
Writing RRS-batch applications using AMI	422
RRS availability	422

Part 7. Setting up an AMI installation 423

Chapter 17. Installation and sample programs 425

Prerequisites	425
Disk space	425
Operating environments	425
MQSeries environment	426
Language compilers	426
Installation on AIX	428
Installation	428
Setting the runtime environment	429
Directory structure (AIX)	430
Installation on AS/400	432
Setting the runtime environment for Java programs	432
Directory structure (AS/400)	433
Installation on HP-UX	436
Installation	436
Setting the runtime environment	437
Directory structure (HP-UX)	438
Installation on OS/390	440
Installation	440
Setting the runtime environment	440
Unicode character conversion	440
Directory structure (OS/390)	441
Installation on Sun Solaris	443
Installation	443

Setting the runtime environment	444
Directory structure (Solaris)	445
Installation on Windows	447
Installation	447
Setting the runtime environment	447
Directory structure (Windows)	448
Local host and repository files (AS/400, UNIX, and Windows)	450
Default location	450
Default names	450
Overriding the default location and names	450
Local host file	451
Repository file	452
Local host and repository files (OS/390)	453
Batch, RRS-batch, IMS	453
CICS	453
Local host file	453
Repository file	454
Repository and local host caches	454
The administration tool	457
Installation	457
Operation	457
Connecting to MQSeries	458
Using MQSeries Integrator Version 1	458
Using MQSeries Publish/Subscribe	458
Using MQSeries Integrator Version 2	458
Migrating to MQSeries Integrator V2 from V1 and MQSeries Publish/Subscribe	459
Creating default MQSeries objects	460
The sample programs	461
Sample programs for AS/400, UNIX, and Windows	461
Running the AS/400, UNIX, and Windows sample programs	462
Sample programs for OS/390	464
Running the sample programs (OS/390)	465

Chapter 18. Defining services and policies 469

Services and policies	469
System provided definitions	470
System default objects	470
Service definitions	472
Service point (sender/receiver)	472
Distribution list	474
Subscriber	474
Publisher	474
Policy definitions	475
Initialization attributes	475
General attributes	476
Send attributes	476
Receive attributes	478
Subscribe attributes	479
Publish attributes	480

Chapter 19. Problem determination 481

Using trace (AS/400, UNIX, and Windows)	481
Trace filename and directory	482
C++ and Java	484
Example trace	484

Using trace (OS/390)	489
Formatted Trace	489
Control of formatted trace	489
GTF Trace	490
Control of GTF Trace	490
When your AMI program fails	492
Reason Codes	492
First failure symptom report (AS/400, UNIX, and Windows)	492
First failure symptom report (OS/390)	492
Other sources of information	493
Common causes of problems	493

Part 8. Appendixes 495

Appendix A. Reason codes 497

Reason code: OK	497
Reason code: Warning	497
Reason code: Failed	499

Appendix B. Constants 509

The constants	509
AMB (Boolean constants)	509
AMBRW (Browse constants)	509
AMCC (Completion codes)	509
AMDEF (Service and policy definitions)	509
AMDT (Definition type constants)	509
AMENC (Encoding constants)	510
AMFB (Feedback codes)	510
AMFMT (Format constants)	510
AMGF and AMGRP (Group status constants)	510

AMH (Handle constants)	510
AMLEN (String length constants)	510
AMMCD (Message Content Descriptor tag names)	510
AMMT (Message types)	510
AMPS (Publish/subscribe)	511
AMRC (Reason codes)	512
AMSD (System default names and handle synonyms)	515
AMWT (Wait time constant)	515

Appendix C. Notices 517

Trademarks	518
----------------------	-----

Glossary of terms and abbreviations 519

Bibliography 521

MQSeries cross-platform publications	521
MQSeries platform-specific publications	521
Softcopy books	522
HTML format	522
Portable Document Format (PDF)	522
BookManager [®] format	523
PostScript format	523
Windows Help format	523
MQSeries information available on the Internet	523

Index 525

Sending your comments to IBM 537

Figures

1. Basic AMI model	7
------------------------------	---

Tables

1. System default objects	14	8. System default objects	470
2. Object interface calls used by the high-level functions	74	9. Service point (sender/receiver).	472
3. System default objects	228	10. Distribution list	474
4. Object interface calls used by the high-level functions	280	11. Subscriber	474
5. The sample programs for AS/400, UNIX, and Windows platforms	461	12. Publisher	474
6. The sample programs for OS/390 ('batch' includes RRS-batch)	464	13. Initialization attributes	475
7. System provided definitions	470	14. General attributes	476
		15. Send attributes	476
		16. Receive attributes	478
		17. Subscribe attributes	479
		18. Publish attributes	480

About this book

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

Who this book is for

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

What you need to know to understand this book

- Knowledge of the C, COBOL, C++, or Java™ programming language is assumed.
- You don't need previous experience of MQSeries to use the Application Messaging Interface (AMI). You can use the examples and sample programs provided to find out how to send and receive messages. However, to understand all the functions of the AMI you need to have some knowledge of the MQSeries Message Queue Interface (MQI). This is described in the *MQSeries Application Programming Guide* and the *MQSeries Application Programming Reference* book.
- You will need to read the following:
 - *MQSeries Publish/Subscribe User's Guide* if you are going to use the AMI with MQSeries Publish/Subscribe.
 - *MQSeries Integrator Version 1.1 Application Development Guide* if you are going to use the AMI with MQSeries Integrator Version 1.1.
 - *MQSeries Integrator Version 2.0 Programming Guide* if you are going to use the AMI with MQSeries Integrator Version 2.0.
- If you are a systems administrator responsible for setting up an installation of the AMI, you need to be experienced in using the MQI.

Structure of this book

This book contains the following parts:

- "Part 1. Introduction" on page 1 gives an overview of the Application Messaging Interface.
- "Part 2. The C interface" on page 9 describes how to use the AMI in C programs. If you are new to MQSeries, gain some experience with the high-level interface first. It provides most of the functionality you need when writing applications. Then move on to the object interface if you need extra functionality.
- "Part 3. The C++ interface" on page 145 describes how to use the AMI in C++ programs.
- "Part 4. The COBOL interface" on page 223 describes how to write AMI programs using the COBOL high-level and object interfaces.
- "Part 5. The Java interface" on page 351 describes how to use the AMI in Java programs.
- "Part 6. OS/390 Subsystems" on page 419 gives advice on writing AMI applications for OS/390® subsystems.

About this book

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

Appearance of text in this book

This book uses the following type styles:

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

amInitialize

The name of an AMI function or method

AMB_TRUE

The name of an AMI constant

AmString getName();

The syntax of AMI functions and methods, and example code

Summary of changes

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

Changes for this edition (SC34-5604-05)

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

- Updates to describe new support for the AS/400[®] system. There are minor changes throughout this manual, and more significant changes in the following sections:
 - “C applications on AS/400” on page 31
 - “C++ applications on AS/400” on page 162
 - “Building Java applications” on page 366
 - “Installation on AS/400” on page 432
 - “Local host and repository files (AS/400, UNIX, and Windows)” on page 450
 - “Using trace (AS/400, UNIX, and Windows)” on page 481
- Editorial changes to clarify the usage notes (and some syntax) for the following calls:
 - “amBrowseMsg” on page 42 (C)
 - “amReceiveMsg” on page 49 (C)
 - “amReceiveRequest” on page 53 (C)
 - “amRcvBrowse” on page 115 (C)
 - “amRcvBrowseSelect” on page 117 (C)
 - “amRcvReceive” on page 122 (C)
 - “AMHBRMS (browse message)” on page 249 (COBOL)
 - “AMHRCMS (receive message)” on page 256 (COBOL)
 - “AMHRCRQ (receive request)” on page 260 (COBOL)
 - “AMRCBR (browse)” on page 322 (COBOL)
 - “AMRCBRSE (browse selection message)” on page 324 (COBOL)
 - “AMRCRC (receive)” on page 329 (COBOL)
- Editorial changes to clarify the syntax of the following calls:
 - “amRcvReceiveFile” on page 124 (C)
 - “AmReceiver” on page 201 (C++)
 - “AmDistributionList” on page 205 (C++)
 - “AmReceiver” on page 399 (Java)
- Editorial changes to clarify the following sections:
 - “Sample programs for AS/400, UNIX, and Windows” on page 461
 - “Service definitions” on page 472
 - “Send attributes” on page 476
 - “Receive attributes” on page 478
 - “Subscribe attributes” on page 479

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

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

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

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

This edition was not published.

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

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

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

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

Part 1. Introduction

Chapter 1. Introduction	3
Main components of the AMI.	3
Sending and receiving messages.	3
Interoperability	3
Programming languages	4
Description of the AMI.	4
Messages	4
Services	4
Point-to-point and publish/subscribe	5
Types of service	5
Policies	6
Application Messaging Interface model	7
Further information	8

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:

<i>MessageID</i>	An identifier for the message. It is usually unique, and typically it is generated by the message transport (MQSeries).
<i>CorrelID</i>	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 <i>MessageID</i> from the request message.
<i>Format</i>	The structure of the message.
<i>Topic</i>	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,

this interface is complex. When using the AMI, these parameters are defined in a service that the systems administrator sets up, so the complexity is hidden from the application programmer.

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

Point-to-point and publish/subscribe

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

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

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

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

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

Types of service

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

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

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

Description of the AMI

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

Policies

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

- The attributes of the message, for example, the priority
- Options for send and receive operations, for example, whether an operation is part of a unit of work
- Publish/subscribe options, for example, whether a publication is retained
- Added value functions that can be invoked as part of the call, such as retry

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

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

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

Application Messaging Interface model

Figure 1 shows the components of the Application Messaging Interface.

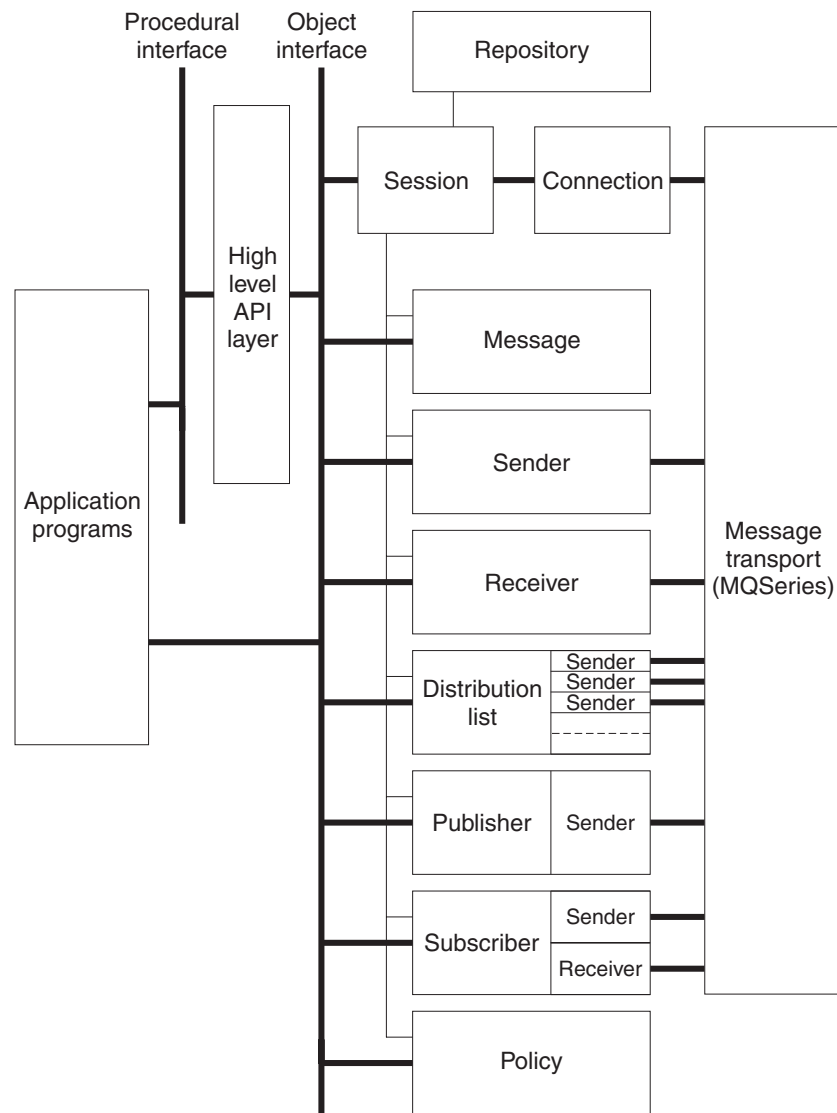


Figure 1. Basic AMI model

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

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

Message, service, and policy objects are created and managed by a session object, which provides the scope for a unit of work. The session object contains a connection object that is not visible to the application. The combination of

Application Messaging Interface model

connection, sender, and receiver objects provides the transport for the message. Other objects, such as helper classes, are provided in C++ and Java.

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

Further information

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

- “Part 2. The C interface” on page 9
- “Part 3. The C++ interface” on page 145
- “Part 4. The COBOL interface” on page 223
- “Part 5. The Java interface” on page 351

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

In “Part 7. Setting up an AMI installation” on page 423, you can find out how to:

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

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

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

Part 2. The C interface

Chapter 2. Using the Application Messaging

Interface in C	13
Structure of the AMI	13
Using the repository	14
System default objects	14
Writing applications in C	16
Opening and closing a session	16
Sending messages	16
Using the message object	17
Sample programs	18
Receiving messages	18
Using the message object	19
Sample programs	19
Request/response messaging	19
Request	20
Response	20
Sample programs	21
File transfer	21
Publish/subscribe messaging	22
Publish	22
Subscribe	23
Sample programs	24
Using name/value elements	24
Parameters	24
Example	25
Error handling	25
Transaction support	26
Sending group messages	26
Other considerations	27
Multithreading	27
Using MQSeries with the AMI	27
Field limits	28
Using the AMI OAMAS subset	28
Building C applications	29
AMI include file	29
Data types	29
Initial values for structures	29
C applications on AIX	30
Preparing C programs on AIX	30
Running C programs on AIX	30
C applications on AS/400	31
Preparing C programs on AS/400	31
Running C programs on AS/400	31
C applications on HP-UX	32
Preparing C programs on HP-UX	32
Running C programs on HP-UX	32
C applications on OS/390	33
Preparing C programs on OS/390	33
Running C programs on OS/390	33
C applications on Solaris	34
Preparing C programs on Solaris	34
Running C programs on Solaris	34
C applications on Windows	35
Preparing C programs on Windows	35
Running C programs on Windows	35

Chapter 3. The C high-level interface 37

Overview of the C high-level interface	38
Initialize and terminate	38
Sending messages	38
Receiving messages	38
File transfer	38
Publish/subscribe	38
Transaction support	38
Reference information for the C high-level interface	39
amBackout	40
amBegin	41
amBrowseMsg	42
amCommit	44
amInitialize	45
amPublish	46
amReceiveFile	47
amReceiveMsg	49
amReceivePublication	51
amReceiveRequest	53
amSendFile	55
amSendMsg	56
amSendRequest	57
amSendResponse	58
amSubscribe	59
amTerminate	60
amUnsubscribe	61

Chapter 4. C object interface overview 63

Session interface functions	64
Session management	64
Create objects	64
Get object handles	64
Delete objects	65
Transactional processing	65
Error handling	65
Message interface functions	66
Get values	66
Set values	66
Reset values	66
Read and write data	66
Publish/subscribe topics	67
Publish/subscribe filters	67
Publish/subscribe name/value elements	67
Error handling	67
Publish/subscribe helper macros	67
Sender interface functions	68
Open and close	68
Send	68
Get values	68
Error handling	68
Receiver interface functions	69
Open and close	69
Receive and browse	69
Get values	69
Set values	69
Error handling	69

Distribution list interface functions	70	amMsgDeleteFilter	93
Open and close	70	Parameters	93
Send.	70	amMsgDeleteNamedElement	93
Get values.	70	amMsgDeleteTopic	94
Error handling	70	amMsgGetCCSID	94
Publisher interface functions.	71	amMsgGetCorrelId	94
Open and close	71	amMsgGetDataLength.	95
Publish	71	amMsgGetDataOffset	95
Get values.	71	amMsgGetElement	95
Error handling	71	amMsgGetElementCCSID.	96
Subscriber interface functions	72	amMsgGetElementCount	96
Open and close	72	amMsgGetEncoding	96
Broker messages.	72	amMsgGetFilter	97
Get values.	72	Parameters	97
Set value	72	amMsgGetFilterCount	97
Error handling	72	Parameters	97
Policy interface functions.	73	amMsgGetFormat	98
Get values.	73	amMsgGetGroupStatus	98
Set value	73	amMsgGetLastError	99
Error handling	73	amMsgGetMsgId	99
High-level functions	74	amMsgGetName	100
Chapter 5. C object interface reference	77	amMsgGetNamedElement	100
Session interface functions	78	amMsgGetNamedElementCount	101
amSesBegin	78	amMsgGetReportCode	101
amSesClearErrorCodes	78	amMsgGetTopic	102
amSesClose	79	amMsgGetType.	102
amSesCommit	79	amMsgGetTopicCount	103
amSesCreate	79	amMsgReadBytes	103
amSesCreateDistList	80	amMsgReset.	103
amSesCreateMessage	80	amMsgSetCCSID	104
amSesCreatePolicy	80	amMsgSetCorrelId.	104
amSesCreatePublisher	81	amMsgSetDataOffset	104
amSesCreateReceiver	81	amMsgSetElementCCSID	105
amSesCreateSender.	82	amMsgSetEncoding	105
amSesCreateSubscriber	82	amMsgSetFormat	106
amSesDelete	83	amMsgSetGroupStatus	106
amSesDeleteDistList	83	amMsgWriteBytes	107
amSesDeleteMessage	83	Message interface helper macros	108
amSesDeletePolicy	84	AmMsgAddStreamName	108
amSesDeletePublisher	84	AmMsgGetPubTimeStamp	108
amSesDeleteReceiver	84	AmMsgGetStreamName.	109
amSesDeleteSender	85	Sender interface functions	110
amSesDeleteSubscriber	85	amSndClearErrorCodes	110
amSesGetDistListHandle	85	amSndClose	110
amSesGetLastError	86	amSndGetCCSID	111
amSesGetMessageHandle.	86	amSndGetEncoding	111
amSesGetPolicyHandle	87	amSndGetLastError	111
amSesGetPublisherHandle	87	amSndGetName	112
amSesGetReceiverHandle.	87	amSndOpen	112
amSesGetSenderHandle	88	amSndSend	113
amSesGetSubscriberHandle	88	amSndSendFile	114
amSesOpen	88	Parameters	114
amSesRollback	89	Usage notes	114
Message interface functions	90	Receiver interface functions.	115
amMsgAddElement	91	amRcvBrowse	115
amMsgAddFilter	91	Usage notes	116
Parameters	91	amRcvBrowseSelect	117
amMsgAddTopic	92	Usage notes	118
amMsgClearErrorCodes	92	amRcvClearErrorCodes	118
amMsgDeleteElement	92	amRcvClose	119
		amRcvGetDefnType	119

amRcvGetLastError	120
amRcvGetName	120
amRcvGetQueueName	121
amRcvOpen	121
amRcvReceive	122
Usage notes	122
amRcvReceiveFile	124
Usage notes	125
amRcvSetQueueName	125
Distribution list interface functions	126
amDstClearErrorCodes	126
amDstClose	126
amDstGetLastError	127
amDstGetName	127
amDstGetSenderCount	128
amDstGetSenderHandle	128
amDstOpen	128
amDstSend	129
amDstSendFile	130
Parameters	130
Usage notes	130
Publisher interface functions	131
amPubClearErrorCodes	131
amPubClose	131
amPubGetCCSID	131
amPubGetEncoding	132
amPubGetLastError	132
amPubGetName	133
amPubOpen	133
amPubPublish	134
Subscriber interface functions	135
amSubClearErrorCodes	135
amSubClose	135
amSubGetCCSID	135
amSubGetDefnType	136
amSubGetEncoding	136
amSubGetLastError	137
amSubGetName	137
amSubGetQueueName	138
amSubOpen	138
amSubReceive	139
amSubSetQueueName	139
amSubSubscribe	140
amSubUnsubscribe	141
Policy interface functions	142
amPolClearErrorCodes	142
amPolGetLastError	142
amPolGetName	143
amPolGetWaitTime	143
amPolSetWaitTime	143

Chapter 2. Using the Application Messaging Interface in C

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

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

This chapter describes the following:

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

Structure of the AMI

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

Session	Contains the AMI session.
Message	Contains the message data, message ID, correlation ID, and options that are used when sending or receiving a message (most of which come from the policy definition).
Sender	This is a service that represents a destination (such as an MQSeries queue) to which messages are sent.
Receiver	This is a service that represents a source from which messages are received.
Distribution list	Contains a list of sender services to provide a list of destinations.
Publisher	Contains a sender service where the destination is a publish/subscribe broker.
Subscriber	Contains a sender service (to send subscribe and unsubscribe messages to a publish/subscribe broker) and a receiver service (to receive publications from the broker).
Policy	Defines how the message should be handled, including items such as priority, persistence, and whether it is included in a unit of work.

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

Structure of the AMI

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

The high-level interface is described in “Chapter 3. The C high-level interface” on page 37. An overview of the object interface is given in “Chapter 4. C object interface overview” on page 63, with reference information in “Chapter 5. C object interface reference” on page 77.

Using the repository

You can run AMI applications with or without a repository. If you do not have a repository, you can use a system default object (see the next section), or create your own by specifying its name on a function call. It will be created using the appropriate system provided definition (see “System provided definitions” on page 470).

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

System default objects

Table 1. System default objects

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

A set of system default objects is created at session creation time. This removes the overhead of creating the objects from applications using these defaults. The system default objects are available for use from both the high-level and object interfaces in C. They are created using the system provided definitions (see “System provided definitions” on page 470).

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

Structure of the AMI

Constants representing synonyms for handles are also provided for these objects, for use from the object interface (see “Appendix B. Constants” on page 509). Note that the first parameter on a call must be a real handle; you cannot use a synonym in this case (that is why handles are not provided for all the default objects).

Writing applications in C

This section gives a number of examples showing how to use the high-level interface of the AMI, with some extensions using the object interface. Equivalent operations to all high-level functions can be performed using combinations of object interface functions (see “High-level functions” on page 74).

Opening and closing a session

Before using the AMI, you must open a session. You can do this with the following high-level function (page 45):

Opening a session

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

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

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

To close a session, you can use the following high-level function (page 60):

Closing a session

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

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

Sending messages

You can send a datagram (send and forget) message using the high-level **amSendMsg** function (page 56). In the simplest case, all you need to specify is the session handle returned by **amInitialize**, the message data, and the message length. Other parameters are set to NULL, so the default message, sender service, and policy objects are used.

Sending a message using all the defaults

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

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

Sending a message using a specified sender service

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

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

Sending a message using a specified policy

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

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

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

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

Sending a message to a distribution list

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

Using the message object

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

Specifying a message object

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

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

Getting an attribute from a message object

```
hMsg = amSesGetMessageHandle(hSession, mySendMsg, &compCode, &reason);  
  
success = amMsgGetMsgId(hMsg, BUFLen, &MsgIdLen, pMsgId,  
&compCode, &reason);
```

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

Writing applications in C

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

Setting an attribute in a message object

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

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

Look at “Message interface functions” on page 66 to find out what other attributes of the message object you can get and set.

After a message object has been used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see **amMsgReset** on page 103) and rebuild it each time.

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

Sample programs

For more details, refer to the *amtshsnd.c* and *amtsosnd.c* sample programs (see “Sample programs for AS/400, UNIX, and Windows” on page 461).

Receiving messages

Use the **amReceiveMsg** high-level function (page 49) to receive a message to which no response is to be sent (such as a datagram). In the simplest case, all you need to specify are the session handle and a buffer for the message data. Other parameters are set to NULL, so the default message, receiver service, and policy objects are used.

Receiving a message using all the defaults

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

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

Receiving a message using a specified receiver service

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

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

Receiving a message using a specified policy

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

The policy can specify, for example:

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

Using the message object

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

Getting an attribute from a message object

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

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

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

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

Using a selection message object

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

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

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

Sample programs

For more details, refer to the `amtshrcv.c` and `amtsorcv.c` sample programs (see “Sample programs for AS/400, UNIX, and Windows” on page 461).

Request/response messaging

In the *request/response* style of messaging, a requester (or client) application sends a request message and expects to receive a message in response. The responder (or server) application receives the request message and produces the response

Writing applications in C

message (or messages) which it returns to the requester application. The responder application uses information in the request message to determine how to send the response message to the requester.

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

Request

Use the **amSendRequest** high-level function (page 57) to send a request message. This is similar to **amSendMsg**, but it includes the name of the service to which the response message is to be sent. In this example the sender service (`mySender`) is specified in addition to the receiver service (`myReceiver`). (A policy name and a send message name can be specified as well, as described in “Sending messages” on page 16.)

Sending a request message

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

The **amReceiveRequest** high-level function (page 53) is used by the responding (or server) application to receive a request message. It is similar to **amReceiveMsg**, but it includes the name of the sender service that will be used for sending the response message. When the message is received, the sender service is updated with the information needed for sending the response to the required destination.

Receiving a request message

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

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

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

Response

After the requested actions have been performed, the responding application sends the response message (or messages) with the **amSendResponse** function (page 58):

Sending a response message

```
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 461).

File transfer

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

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

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

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

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

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

If the message selected for the receive operation does not contain file information, it is returned to the application in the message object named on the call and a warning is returned with reason `AMRC_NOT_A_FILE`. If the file transfer fails part way through a message, that message is returned to the application and the

Writing applications in C

current data pointer within the message shows how far it had been processed before the error occurred. Again, we recommend the use of the policy syncpoint option and checking of completion and reason codes to ensure the whole file was received correctly before committing the unit of work. If the file was sent from a different type of file system than it is received into, the AMI converts the file and returns a warning with reason `AMRC_FILE_FORMAT_CONVERTED`. This conversion allows transfer between OS/390 datasets with different record types or sizes, and between OS/390 datasets and the flat files used on other systems.

If the intent is not to transfer a file from one location to another, but rather to generate a group of messages from a file, you should use the 'logical splitting' policy option. If the message object referenced by the send call has a format of `AMFMT_STRING`, the file is split into lines and each line is sent as a separate message. Any other format indicates that the file does not contain text. If the record length of a non-text file is known (as in the case of OS/390 datasets) then each record is sent as a separate message. If the record length of a non-text file is not known then the whole file is considered to be a single record, and is sent in one message. No extra header information is added to the file data. The messages can then be processed in the same fashion as any other message in your queueing network.

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

Publish/subscribe messaging

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

Subscribing applications can exploit content-based publish/subscribe by passing a filter on subscribe and unsubscribe calls (see "Using MQSeries Integrator Version 2" on page 458).

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

Publish

Use the `amPublish` high-level function (page 46) to publish a message. You need to specify the name of the publisher for the publish/subscribe broker. The topic relating to this publication and the publication data must also be specified:

Publishing a message

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


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

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

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

Subscribe

The `amSubscribe` high-level function (page 59) is used to subscribe to a publish/subscribe broker specified by the name of a subscriber service. The receiver to which publications will be sent is included within the definition of the subscriber. The name of a receiver service to which the broker can send a response message (`myReceiver`) is also specified.

Subscribing to a broker

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

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

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

To remove a subscription, use the `amUnsubscribe` high-level function (page 61). To remove all subscriptions, you can specify a policy that has the 'Deregister All Topics' subscriber attribute.

To receive a publication from a broker, use the `amReceivePublication` function (page 51). For example:

Receiving a publication

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

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

Writing applications in C

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

Sample programs

For more details, refer to the `amtshpub.c`, `amtshsub.c`, `amtsopub.c`, and `amtsosub.c` sample programs (see “Sample programs for AS/400, UNIX, and Windows” on page 461).

Using name/value elements

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

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

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

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

Parameters

strucId	The AMELEM structure identifier (input). Its value must be <code>AMELEM_STRUC_ID</code> . The constant <code>AMELEM_STRUC_ID_ARRAY</code> is also defined; this has the same value as <code>AMELEM_STRUC_ID</code> but is an array of characters instead of a string.
version	The version number of the AMELEM structure (input). Its value must be <code>AMELEM_VERSION_1</code> .
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 <code>nameBuffLen</code> parameter value is set to 0, the AMI returns the <code>nameLen</code> value but not the <code>pName</code> value. This is not an error.
nameLen	The length of the name in bytes (input or output). A value of <code>AMLEN_NULL_TERM</code> denotes a null-terminated string of unspecified length.

pName	The name buffer (input or output).
valueBuffLen	The length of the value buffer (input). If <code>valueBuffLen</code> is set to zero, the AMI returns the <code>valueLen</code> value but not the <code>pValue</code> value. This is not an error.
valueLen	The value length in bytes (input or output). A value of <code>AMLEN_NULL_TERM</code> denotes a null-terminated string of unspecified length.
pValue	The value buffer (input or output).
typeBuffLen	Reserved, must be zero.
typeLen	Reserved, must be zero.
pType	Reserved, must be NULL.

Example

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

```
pName  AMPS_COMMAND
pValue AMPS_REQUEST_UPDATE
```

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

Using name/value elements

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

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

If you need to use streams with 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.

Writing applications in C

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

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

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

Transaction support

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

The API calls used to control the transaction 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
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.

Writing applications in C

Field limits

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

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)
hlq.SCSQC370	(OS/390)
/amt/inc	(UNIX)
\amt\include	(Windows)

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

Your AMI C program must contain the statement:

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

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

Data types

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

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

Initial values for structures

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

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

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

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

Building C applications

Next step

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

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

C applications on AIX

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

Preparing C programs on AIX

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

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

- Where the AMI include files are.
To do this, use the `-I` flag. In the case of AIX, they are usually located at `/usr/mqm/amt/inc`.
- Where the AMI library is.
To do this, use the `-L` flag. In the case of AIX, it is usually located at `/usr/mqm/lib`.
- Link with the AMI library.
To do this, use the `-l` flag, more specifically `-lamt`.

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

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

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

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

Running C programs on AIX

To run a C executable, you must have access to the C libraries `libamt.a`, `libamtXML310.a`, and `libamtICUUC140.a` in your runtime environment. If the `amtInstall` utility has been run, this environment will be set up for you (see “Installation on AIX” on page 428).

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

To do this:

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

You must have sufficient access to perform this operation.

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

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

You must also make the AMI 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/mqm/lib/amtcmqm /usr/lib/amtcmqm
```

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 426 for compilers supported by the AMI.

Preparing C programs on AS/400

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

To compile a C module, you can use the OS/400® command **CRTCMOD**. The library **QMQMAMI** must be in the library list because it contains the **amt.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) BDNSRVPGM(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 **BDNSRVPGM** parameter of **CRTPGM**. At run time, **QMQMAMI** must be in the library list.

C applications on HP-UX

C applications on HP-UX

This section explains what you have to do to prepare and run your C programs on the HP-UX operating system. See “Language compilers” on page 426 for compilers supported by the AMI.

Preparing C programs on HP-UX

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

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

- Where the AMI include files are.
To do this, use the **-I** flag. In the case of HP-UX, they are usually located at `/opt/mqm/amt/inc`.
- Where the AMI libraries are.
To do this, use the **-Wl,+b,,-L** flags. In the case of HP-UX, they are usually located at `/opt/mqm/lib`.
- Link with the AMI library.
To do this, use the **-l** flag, more specifically **-lamt**.

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

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

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

Running C programs on HP-UX

To run a C executable, you must have access to the C libraries `libamt.sl`, `libamtXML310.sl`, and `libamtICUUC140.sl` in your runtime environment. If the **amtInstall** utility has been run, this environment will be set up for you (see “Installation on HP-UX” on page 436).

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

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

You must have sufficient access to perform this operation.

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

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

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

For the non-threaded MQSeries Server library, perform:

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

For the non-threaded MQSeries Client library, perform:

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

For the threaded MQSeries Server library, perform:

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

For the threaded MQSeries Client library, perform:

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

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

C applications on OS/390

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

Preparing C programs on OS/390

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

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

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

Batch	AMTBD10
RRS-batch	AMTRD10
CICS	AMTCD10
IMS	AMTID10

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

Running C programs on OS/390

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

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

C applications on Solaris

C applications on Solaris

This section explains what you have to do to prepare and run your C programs in the Sun Solaris operating environment. See “Language compilers” on page 426 for compilers supported by the AMI.

Preparing C programs on Solaris

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

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

- Where the AMI include files are.
To do this, use the `-I` flag. In the case of Solaris, they are usually located at `/opt/mqm/amt/inc`.
- Where the AMI library is.
To do this, use the `-L` flag. In the case of Solaris, it is usually located at `/opt/mqm/lib`.
- Link with the AMI library.
To do this, use the `-l` flag, more specifically `-lamt`.

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

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

Running C programs on Solaris

To run a C executable, you must have access to the C libraries `libamt.so`, `libamtXML310.so`, and `libamtICUUC140.so` in your runtime environment. If the `amtInstall` utility has been run, this environment will be set up for you (see “Installation on Sun Solaris” on page 443).

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

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

You must have sufficient access to perform this operation.

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

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

For the MQSeries Client library, perform:

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

C applications on Windows

This section explains what you have to do to prepare and run your C programs on the Windows 98 and Windows NT operating systems. See “Language compilers” on page 426 for compilers supported by the AMI.

Preparing C programs on Windows

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

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

- Where the AMI include files are.

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

- Where the AMI library is.

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

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

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

Running C programs on Windows

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

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

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

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

C applications on Windows

Chapter 3. The C high-level interface

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

This chapter contains:

- “Overview of the C high-level interface” on page 38
- “Reference information for the C high-level interface” on page 39

Overview of the C high-level interface

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

Initialize and terminate

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

amInitialize	page 45
amTerminate	page 60

Sending messages

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

amSendMsg	page 56
amSendRequest	page 57
amSendResponse	page 58

Receiving messages

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

amReceiveMsg	page 49
amReceiveRequest	page 53
amBrowseMsg	page 42

File transfer

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

amSendFile	page 55
amReceiveFile	page 47

Publish/subscribe

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

amPublish	page 46
amSubscribe	page 59
amUnsubscribe	page 61
amReceivePublication	page 51

Transaction support

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

amBegin	page 41
amCommit	page 44
amBackout	page 40

Reference information for the C high-level interface

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

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

If the completion code returns warning or failed, the reason code identifies the reason for the error or warning (see “Appendix A. Reason codes” on page 497).

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

amBackout

Function to back out a unit of work.

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

Parameters

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

amBegin

Function to begin a unit of work.

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

Parameters

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

amBrowseMsg

Function to browse a message. See the *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     rcvMsgName,
    AMSTR     senderName,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

Parameters

hSession	The session handle returned by amInitialize (input).
receiverName	The name of a receiver service (input). If specified as NULL, the system default receiver name (constant: AMSD_RCV) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
options	Options controlling the browse operation (input). Possible values are: <pre>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</pre> <p>AMBRW_RECEIVE_CURRENT is equivalent to amRcvReceive for the message under the browse cursor.</p> <p>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.</p>
buffLen	The length in bytes of a buffer in which the data is returned (input).
pDataLen	The length of the message data, in bytes (output). Specify as NULL if this is not required.
pData	The received message data (output).
rcvMsgName	The name of the message object for the received message (output). Properties, and message data if not returned in the pData parameter, can be extracted from the message object using the object interface (see "Message interface functions" on page 90). The message object is implicitly reset before the browse takes place. If rcvMsgName is specified as NULL, the system default receive message name (constant: AMSD_RCV_MSG) is used.

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

Usage notes

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

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

To return data in an application message buffer:

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

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

To return only the data length:

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

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

amCommit

Function to commit a unit of work.

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

Parameters

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

amInitialize

Function to create and open an AMI session. It returns a session handle of type `AMHSES`, which is valid until the session is terminated. One `amInitialize` is allowed per thread. A session handle can be used on different threads, subject to any limitations of the underlying transport layer (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 <code>NULL</code> , the system default policy name (constant: <code>AMSD_POL</code>) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amPublish

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

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

Parameters

hSession	The session handle returned by amInitialize (input).
publisherName	The name of a publisher service (input). If specified as NULL, the system default publisher name (constant: AMSD_PUB) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
responseName	The name of the receiver service to which the response to this publish request should be sent (input). Specify as NULL if no response is required. This parameter is mandatory if the policy specifies implicit publisher registration (the default).
topicLen	The length of the topic for this publication, in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is NULL terminated.
pTopic	The topic for this publication (input).
dataLen	The length of the publication data in bytes (input). A value of zero indicates that any publication data has been added to the message object (pubMsgName) using the object interface (see “Message interface functions” on page 90).
pData	The publication data, if dataLen is non-zero (input).
pubMsgName	The name of a message object that contains the header for the publication message (input). If dataLen is zero, it also holds any publication data. If specified as NULL, the system default message name (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amReceiveFile

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

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

Parameters

hSession	The session handle returned by amInitialize (input).
receiverName	The name of a receiver service (input). If specified as NULL, the system default receiver name (constant: AMSD_RCV) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
options	A reserved field that must be specified as zero (input).
selMsgName	Optional selection message object used to specify information (such as a <i>CorrelId</i>) needed to select the required message (input).
directoryLen	A reserved field that must be specified as zero (input).
directory	A reserved field that must be specified as NULL (input).
fileNameLen	The length of the file name in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
fileName	The name of the file into which the transferred data is to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If NULL or a null string is specified, the AMI will use the name of the originating file (including any directory prefix), exactly as it was supplied on the send file call. Note that the original file name may not be appropriate for use by the receiver, either because a path name included in the file name is not applicable to the receiving system, or because the sending and receiving systems use different file name conventions.
rcvMsgName	The name of the message object to be used to receive the file (output). This parameter is updated with the message properties (for example, the Message ID). If the message is not from a file, rcvMsgName receives the message data. If specified as NULL, the system default receive message name (constant AMSD_RCV_MSG) is used. Property information and message data can be extracted from the message object using the object interface (see “Message interface functions” on page 90). The message object is reset implicitly before the receive takes place.
pCompCode	Completion code (output).

C high-level interface

pReason Reason code (output).

Usage notes

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

amReceiveMsg

Function to receive a message.

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

Parameters

hSession	The session handle returned by amInitialize (input).
receiverName	The name of a receiver service (input). If specified as NULL, the system default receiver name (constant: AMSD_RCV) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
selMsgName	Optional selection message object used to specify information (such as a <i>CorrelId</i>) 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 <i>pData</i> parameter, can be extracted from the message object using the object interface (see “Message interface functions” on page 90). The message object is implicitly reset before the receive takes place.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

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 <i>CorrelId</i>) 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 (<i>rcvMsgName</i>) using the object interface (see “Message interface functions” on page 90).
pDataLen	The length in bytes of the publication data (output). Specify as NULL if this is not required.
pData	The publication data (output). Specify as NULL if this is not required. Data can be extracted from the message object (<i>rcvMsgName</i>) using the object interface (see “Message interface functions” on page 90).
rcvMsgName	The name of a message object for the received message (input). If specified as NULL, the default message name (constant: AMSD_RCV_MSG) is used. The publication message properties and data update this message object, in addition to being returned in the parameters above. The message object is implicitly reset to the default before the receive takes place.

C high-level interface

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

Usage notes

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

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

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

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

amReceiveRequest

Function to receive a request message.

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

Parameters

hSession	The session handle returned by amInitialize (input).
receiverName	The name of a receiver service (input). If specified as NULL, the system default receiver name (constant: AMSD_RCV) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
buffLen	The length in bytes of a buffer in which the data is returned (input).
pDataLen	The length of the message data, in bytes (output). Specify as NULL if this is not required.
pData	The received message data (output).
rcvMsgName	The name of the message object for the received message (output). If specified as NULL, the system default receiver service (constant: AMSD_RCV_MSG) is used. Header information, and message data if not returned in the Data parameter, can be extracted from the message object using the object interface (see “Message interface functions” on page 90). The message object is implicitly reset before the receive takes place.
senderName	The name of a special type of sender service known as a <i>response sender</i> , to which the response message will be sent (output). This sender name must not be defined in the repository. If specified as NULL, the system default response sender service (constant: AMSD_RSP_SND) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

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

Data conversion

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

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

C high-level interface

Use of the buffLen parameter

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

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

To return data in an application message buffer:

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

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

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

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

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

To remove the message from the queue and discard it:

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

The message will be discarded with an AMRC_MSG_TRUNCATED warning.

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

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

amSendFile

Function to send data from a file.

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

Parameters

hSession	The session handle returned by amInitialize (input).
senderName	The name of a sender service (input). If specified as NULL, the system default sender name (constant: AMSD_SND) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
options	A reserved field that must be specified as zero (input).
directoryLen	A reserved field that must be specified as zero (input).
directory	A reserved field that must be specified as NULL (input).
fileNameLen	The length of the file name in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
fileName	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with a receive file call (see “amReceiveFile” on page 47 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
sndMsgName	The name of the message object to be used to send the file (input). This parameter can be used, for example, to specify the Correlation ID, which can be set from the message object using the object interface (see “Message interface functions” on page 90).
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

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

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

amSendMsg

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

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

Parameters

hSession	The session handle returned by amInitialize (input).
senderName	The name of a sender service (input). If specified as NULL, the system default sender name (constant: AMSD_SND) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy name (constant: AMSD_POL) is used.
dataLen	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (sndMsgName) using the object interface (see “Message interface functions” on page 90).
pData	The message data, if dataLen is non-zero (input).
sndMsgName	The name of a message object for the message being sent (input). If dataLen is zero it also holds any message data. If specified as NULL, the system default message name (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSendRequest

Function to send a request message.

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

Parameters

hSession	The session handle returned by amInitialize (input).
senderName	The name of a sender service (input). If specified as NULL, the system default sender name (constant: AMSD_SND) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy (constant: AMSD_POL) is used.
responseName	The name of the receiver service to which the response to this send request should be sent (input). See amReceiveRequest . Specify as NULL if no response is required.
dataLen	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (sndMsgName) using the object interface (see “Message interface functions” on page 90).
pData	The message data, if dataLen is non-zero (input).
sndMsgName	The name of a message object for the message being sent (input). If specified as NULL, the system default message (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSendResponse

Function to send a response to a request message.

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

Parameters

hSession	The session handle returned by amInitialize (input).
senderName	The name of the sender service (input). It must be set to the senderName specified for the amReceiveRequest function.
policyName	The name of a policy (input). If specified as NULL, the system default policy (constant: AMSD_POL) is used.
rcvMsgName	The name of the received message that this message is a response to (input). It must be set to the rcvMsgName specified for the amReceiveRequest function.
dataLen	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (sndMsgName) using the object interface (see “Message interface functions” on page 90).
pData	The message data, if dataLen is non-zero (input).
sndMsgName	The name of a message object for the message being sent (input). If specified as NULL, the system default message (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubscribe

Function to register a subscription with a publish/subscribe broker.

Publications matching the subscription are sent to the receiver service associated with the subscriber. By default, this has the same name as the subscriber service, with the addition of the suffix '.RECEIVER'.

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

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

Parameters

hSession	The session handle returned by amInitialize (input).
subscriberName	The name of a subscriber service (input). If specified as NULL, the system default subscriber (constant: AMSD_SUB) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy (constant: AMSD_POL) is used.
responseName	The name of the receiver service to which the response to this subscribe request should be sent (input). Specify as NULL if no response is required. This is not the service to which publications will be sent by the broker; they are sent to the receiver service associated with the subscriber (see above).
topicLen	The length of the topic for this subscription, in bytes (input).
pTopic	The topic for this subscription (input). Publications which match this topic, including wildcards, will be sent to the subscriber. Multiple topics can be specified in the message object (subMsgName) using the object interface (see "Message interface functions" on page 90).
filterLen	The length in bytes of the filter (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
pFilter	The filter to be added (input). The syntax of the filter string is described in the <i>MQSeries Integrator Version 2.0 Programming Guide</i> .
subMsgName	The name of a message object for the subscribe message (input). If specified as NULL, the system default message (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amTerminate

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

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

Parameters

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

amUnsubscribe

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

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

Parameters

hSession	The session handle returned by amInitialize (input).
subscriberName	The name of a subscriber service (input). If specified as NULL, the system default subscriber (constant: AMSD_SUB) is used.
policyName	The name of a policy (input). If specified as NULL, the system default policy (constant: AMSD_POL) is used.
responseName	The name of the receiver service to which the response to this unsubscribe request should be sent (input). Specify as NULL if no response is required.
topicLen	The length of the topic, in bytes (input).
pTopic	The topic that identifies the subscription to be removed (input). Multiple topics can be specified in the message object (unsubMsgName) using the object interface (see “Message interface functions” on page 90). To deregister all topics, a policy that provides this option must be specified (this is not the default policy). Otherwise, to remove a previous subscription, the topic information specified must match that specified on the relevant amSubscribe request.
filterLen	The length in bytes of the filter (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
pFilter	The filter that identifies the subscription to be removed (input). The syntax of the filter string is described in the <i>MQSeries Integrator Version 2.0 Programming Guide</i> .
unsubMsgName	The name of a message object for the unsubscribe message (input). If specified as NULL, the system default message (constant: AMSD_SND_MSG) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

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

Chapter 4. C object interface overview

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

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

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

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

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

Details of the object interface functions used by each high-level function are given on page 74.

Session interface functions

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

Session management

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

amSesCreate	page 79
amSesOpen	page 88
amSesClose	page 79
amSesDelete	page 83

Create objects

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

amSesCreateMessage	page 80
amSesCreateSender	page 82
amSesCreateReceiver	page 81
amSesCreateDistList	page 80
amSesCreatePublisher	page 81
amSesCreateSubscriber	page 82
amSesCreatePolicy	page 80

Get object handles

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

amSesGetMessageHandle	page 86
amSesGetSenderHandle	page 88
amSesGetReceiverHandle	page 87
amSesGetDistListHandle	page 85
amSesGetPublisherHandle	page 87
amSesGetSubscriberHandle	page 88
amSesGetPolicyHandle	page 87

Delete objects

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

amSesDeleteMessage	page 83
amSesDeleteSender	page 85
amSesDeleteReceiver	page 84
amSesDeleteDistList	page 83
amSesDeletePublisher	page 84
amSesDeleteSubscriber	page 85
amSesDeletePolicy	page 84

Transactional processing

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

amSesBegin	page 78
amSesCommit	page 79
amSesRollback	page 89

Error handling

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

amSesClearErrorCodes	page 78
amSesGetLastError	page 86

Message interface functions

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

Get values

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

amMsgGetCCSID	page 94
amMsgGetCorrelId	page 94
amMsgGetElementCCSID	page 96
amMsgGetEncoding	page 96
amMsgGetFormat	page 98
amMsgGetGroupStatus	page 98
amMsgGetMsgId	page 99
amMsgGetName	page 100
amMsgGetReportCode	page 101
amMsgGetType	page 102

Set values

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

amMsgSetCCSID	page 104
amMsgSetCorrelId	page 104
amMsgSetElementCCSID	page 105
amMsgSetEncoding	page 105
amMsgSetFormat	page 106
amMsgSetGroupStatus	page 106

Reset values

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

amMsgReset	page 103
-------------------	----------

Read and write data

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

amMsgGetDataLength	page 95
amMsgGetDataOffset	page 95
amMsgSetDataOffset	page 104
amMsgReadBytes	page 103
amMsgWriteBytes	page 107

Publish/subscribe topics

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

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

Publish/subscribe filters

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

amMsgAddFilter	page 91
amMsgDeleteFilter	page 93
amMsgGetFilter	page 97
amMsgGetFilterCount	page 97

Publish/subscribe name/value elements

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

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

Error handling

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

amMsgClearErrorCodes	page 92
amMsgGetLastError	page 99

Publish/subscribe helper macros

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

AmMsgAddStreamName	page 108
AmMsgGetPubTimestamp	page 108
AmMsgGetStreamName	page 109

Sender interface functions

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

Open and close

Functions to open and close the sender service.

amSndOpen	page 112
amSndClose	page 110

Send

Function to send a message.

amSndSend	page 113
amSndSendFile	page 114

Get values

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

amSndGetCCSID	page 111
amSndGetEncoding	page 111
amSndGetName	page 112

Error handling

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

amSndClearErrorCodes	page 110
amSndGetLastError	page 111

Receiver interface functions

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

Open and close

Functions to open and close the receiver service.

amRcvOpen	page 121
amRcvClose	page 119

Receive and browse

Functions to receive or browse a message.

amRcvReceive	page 122
amRcvReceiveFile	page 124
amRcvBrowse	page 115
amRcvBrowseSelect	page 117

Get values

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

amRcvGetDefnType	page 119
amRcvGetName	page 120
amRcvGetQueueName	page 121

Set values

Function to set the queue name of the receiver service.

amRcvSetQueueName	page 125
--------------------------	----------

Error handling

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

amRcvClearErrorCodes	page 118
amRcvGetLastError	page 120

Distribution list interface functions

A distribution list object encapsulates a list of sender services.

Open and close

Functions to open and close the distribution list service.

amDstOpen page 128

amDstClose page 126

Send

Function to send a message to the distribution list.

amDstSend page 129

amDstSendFile page 130

Get values

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

amDstGetName page 127

amDstGetSenderCount page 128

amDstGetSenderHandle page 128

Error handling

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

amDstClearErrorCodes page 126

amDstGetLastError page 127

Publisher interface functions

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

Open and close

Functions to open and close the publisher service.

amPubOpen	page 133
amPubClose	page 131

Publish

Function to publish a message.

amPubPublish	page 134
---------------------	----------

Get values

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

amPubGetCCSID	page 131
amPubGetEncoding	page 132
amPubGetName	page 133

Error handling

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

amPubClearErrorCodes	page 131
amPubGetLastError	page 132

Subscriber interface functions

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

Open and close

Functions to open and close the subscriber service.

amSubOpen	page 138
amSubClose	page 135

Broker messages

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

amSubSubscribe	page 140
amSubUnsubscribe	page 141
amSubReceive	page 139

Get values

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

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

Set value

Function to set the queue name of the subscriber service.

amSubSetQueueName	page 139
--------------------------	----------

Error handling

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

amSubClearErrorCodes	page 135
amSubGetLastError	page 137

Policy interface functions

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

Get values

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

amPolGetName page 143

amPolGetWaitTime page 143

Set value

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

amPolSetWaitTime page 143

Error handling

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

amPolClearErrorCodes page 142

amPolGetLastError page 142

High-level functions

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

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
<p>Note:</p> <p>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.</p>	

C object interface overview

Chapter 5. C object interface reference

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

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

Within each section the functions are listed in alphabetical order.

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

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

If the completion code returns warning or failed, the reason code identifies the reason for the error or warning (see “Appendix A. Reason codes” on page 497).

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

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

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

Session interface functions

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

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

amSesBegin

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

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

hSess The session handle returned by **amSesCreate** (input).

hPolicy The handle of a policy (input). If specified as AMH_NULL_HANDLE, the system default policy (constant: AMSD_POL_HANDLE) is used.

pCompCode Completion code (output).

pReason Reason code (output).

amSesClearErrorCodes

Clears the error codes in the session object.

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

hSess The session handle returned by **amSesCreate** (input).

pCompCode Completion code (output).

pReason Reason code (output).

amSesClose

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

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

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

amSesCommit

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

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

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

amSesCreate

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

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

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

C session interface

amSesCreateDistList

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

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

hSess The session handle returned by **amSesCreate** (input).

name The name of the distribution list (input). This must match the name of a distribution list defined in the repository.

pCompCode Completion code (output).

pReason Reason code (output).

amSesCreateMessage

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

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

hSess The session handle returned by **amSesCreate** (input).

name The name of the message (input). This can be any name that is meaningful to the application. It is specified so that this message object can be used with the high-level interface.

pCompCode Completion code (output).

pReason Reason code (output).

amSesCreatePolicy

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

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

hSess The session handle returned by **amSesCreate** (input).

name The name of the policy (input). If it matches a policy defined in the repository, the policy will be created using the repository definition, otherwise it will be created with default values.

If a repository is being used and the named policy is not found in the repository, a completion code of AMCC_WARNING is returned with a reason code of AMRC_POLICY_NOT_IN_REPOS.

pCompCode Completion code (output).

pReason Reason code (output).

amSesCreatePublisher

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

```

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

```

hSess The session handle returned by **amSesCreate** (input).

name The name of the publisher (input). If it matches a publisher defined in the repository, the publisher will be created using the repository definition, otherwise it will be created with default values (that is, with a sender service name that matches the publisher name).

If a repository is being used and the named publisher is not found in the repository, a completion code of AMCC_WARNING is returned with a reason code of AMRC_PUBLISHER_NOT_IN_REPOS.

pCompCode Completion code (output).

pReason Reason code (output).

amSesCreateReceiver

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

```

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

```

hSess The session handle returned by **amSesCreate** (input).

name The name of the receiver service (input). If it matches a receiver defined in the repository, the receiver will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the receiver name).

If a repository is being used and the named receiver is not found in the repository, a completion code of AMCC_WARNING is returned with a reason code of AMRC_RECEIVER_NOT_IN_REPOS.

pCompCode Completion code (output).

pReason Reason code (output).

C session interface

amSesCreateSender

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

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

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

amSesCreateSubscriber

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

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

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

amSesDelete

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

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

phSess A *pointer* to the session handle returned by **amSesCreate** (input/output).

pCompCode Completion code (output).

pReason Reason code (output).

amSesDeleteDistList

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

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

hSess The session handle returned by **amSesCreate** (input).

phDistList A *pointer* to the distribution list handle (input/output).

pCompCode Completion code (output).

pReason Reason code (output).

amSesDeleteMessage

Deletes a message object.

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

hSess The session handle returned by **amSesCreate** (input).

phMsg A *pointer* to the message handle (input/output).

pCompCode Completion code (output).

pReason Reason code (output).

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).
pReason Reason code (output).

amSesDeleteSender

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

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

hSess The session handle returned by **amSesCreate** (input).

phSender A *pointer* to the sender service handle (input/output).

pCompCode Completion code (output).

pReason Reason code (output).

amSesDeleteSubscriber

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

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

hSess The session handle returned by **amSesCreate** (input).

phSub A *pointer* to the subscriber handle (input/output).

pCompCode Completion code (output).

pReason Reason code (output).

amSesGetDistListHandle

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

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

hSess The session handle returned by the **amSesCreate** function (input).

name The name of the distribution list (input).

pCompCode Completion code (output).

pReason Reason code (output).

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).

pReason Reason code (output).

amSesGetPublisherHandle

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

```

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

```

hSess The session handle returned by **amSesCreate** (input).

name The name of the publisher (input).

pCompCode Completion code (output).

pReason Reason code (output).

amSesGetReceiverHandle

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

```

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

```

hSess The session handle returned by **amSesCreate** (input).

name The name of the receiver service (input).

pCompCode Completion code (output).

pReason Reason code (output).

C session interface

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).

pReason Reason code (output).

amSesRollback

Rolls back a unit of work.

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

hSess The session handle returned by **amSesCreate** (input).

hPolicy The handle of a policy (input). If specified as `AMH_NULL_HANDLE`, the system default policy (constant: `AMSD_POL_HANDLE`) is used.

pCompCode Completion code (output).

pReason Reason code (output).

Message interface functions

A *message* object encapsulates 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:

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

When a message object is used to send a message, it will not normally be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see **amMsgReset** on page 103) and rebuild it each time.

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

amMsgAddElement	page 91
amMsgDeleteElement	page 92
amMsgGetElement	page 95
amMsgGetElementCount	page 96
amMsgDeleteNamedElement	page 93
amMsgGetNamedElement	page 100
amMsgGetNamedElementCount	page 101
amMsgAddTopic	page 92
amMsgDeleteTopic	page 94
amMsgGetTopic	page 102
amMsgGetTopicCount	page 103

amMsgAddElement

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

```

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

```

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

amMsgAddFilter

Adds a filter to a subscribe or unsubscribe request message.

```

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

```

Parameters

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

C message interface

amMsgAddTopic

Adds a topic to a publish/subscribe message.

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

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

topicLen The length in bytes of the topic (input). A value of `AMLEN_NULL_TERM` specifies that the string is NULL terminated.

pTopic The topic to be added (input).

pCompCode Completion code (output).

pReason Reason code (output).

amMsgClearErrorCodes

Clears the error codes in the message object.

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

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

pCompCode Completion code (output).

pReason Reason code (output).

amMsgDeleteElement

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

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

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

elemIndex The index of the required element in the message, starting from zero (input). On completion, elements with higher `elemIndex` values than that specified will have their index value reduced by one.

amMsgGetElementCount gets the number of elements in the message.

pCompCode Completion code (output).

pReason Reason code (output).

amMsgDeleteFilter

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

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

Parameters

hMsg	The message handle returned by <code>amSesCreateMessage</code> (input).
filterIndex	The index of the required filter in the message, starting from zero (input). <code>amMsgGetFilterCount</code> 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 <code>amSesCreateMessage</code> (input).
nameIndex	The index of the required named element in the message (input). Specifying an index of zero deletes the <i>first</i> element with the specified name. On completion, elements with higher <code>nameIndex</code> 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 <code>AMLEN_NULL_TERM</code> specifies that the string is NULL terminated.
pName	The name of the element to be deleted (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

C message interface

amMsgDeleteTopic

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

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

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

topicIndex The index of the required topic in the message, starting from zero (input). **amMsgGetTopicCount** gets the number of topics in the message.

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetCCSID

Gets the coded character set identifier of the message.

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

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

pCCSID The coded character set identifier (output).

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetCorrelId

Gets the correlation identifier of the message.

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

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

buffLen The length in bytes of a buffer in which the correlation identifier is returned (input).

pCorrelIdLen The length of the correlation identifier, in bytes (output). If specified as NULL, the length is not returned.

pCorrelId The correlation identifier (output).

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetDataLength

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

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

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

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

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetDataOffset

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

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

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

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

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetElement

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

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

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

elemIndex The index of the required element in the message, starting from zero (input). **amMsgGetElementCount** gets the number of elements in the message.

pElem The selected element in the message (output).

pCompCode Completion code (output).

pReason Reason code (output).

C message interface

amMsgGetElementCCSID

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

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

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

pElementCCSID The element coded character set identifier (output).

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetElementCount

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

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

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

pCount The number of elements in the message (output).

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetEncoding

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

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

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

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

```
AMENC_NATIVE  
AMENC_NORMAL  
AMENC_NORMAL_FLOAT_390  
AMENC_REVERSED  
AMENC_REVERSED_FLOAT_390  
AMENC_UNDEFINED
```

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetFilter

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

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

Parameters

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

amMsgGetFilterCount

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

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

Parameters

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

C message interface

amMsgGetFormat

Gets the format of the message.

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

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

amMsgGetGroupStatus

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

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

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

amMsgGetLastError

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

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

hMsg	The message handle returned by amSesCreateMessage (input).
buffLen	Reserved, must be zero (input).
pStringLen	Reserved, must be NULL (input).
pErrorText	Reserved, must be NULL (input).
pReason2	A secondary reason code (output). Not returned if specified as NULL. If pReason indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, pReason2 gives 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.

amMsgGetMsgId

Gets the message identifier.

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

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

C message interface

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 <code>AMLEN_NULL_TERM</code> specifies that the string is null terminated.
pName	The element name (input).
pElem	The selected named element in the message (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetNamedElementCount

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

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

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

nameLen The length of the element name, in bytes (input). A value of `AMLEN_NULL_TERM` specifies that the string is null terminated.

pName The specified element name (input).

pCount The number of elements in the message with the specified name (output).

pCompCode Completion code (output).

pReason Reason code (output).

amMsgGetReportCode

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

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

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

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

- AMFB_EXPIRATION
- AMFB_COA
- AMFB_COD
- AMFB_ERROR

pCompCode Completion code (output).

pReason Reason code (output).

C message interface

amMsgGetTopic

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

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

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

amMsgGetType

Gets the message type from a message.

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

hMsg	The message handle returned by amSesCreateMessage (input).
PType	The message type (output). The following values can be returned: AMMT_DATAGRAM AMMT_REQUEST AMMT_REPLY AMMT_REPORT
pCompCode	Completion code (output).
pReason	Reason code (output).

amMsgGetTopicCount

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

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

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

pCount The number of topics (output).

pCompCode Completion code (output).

pReason Reason code (output).

amMsgReadBytes

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

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

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

readLen The maximum number of bytes to be read (input). The data buffer specified by **pData** must be at least this size. The number of bytes returned is the minimum of **readLen** and the number of bytes between the data offset and the end of the data.

pBytesRead The number of bytes read (output). If specified as **NULL**, the number is not returned.

pData The read data (output).

pCompCode Completion code (output).

pReason Reason code (output).

amMsgReset

Resets the message object its initial state (see page 90).

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

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

options A reserved field that must be specified as zero (input).

pCompCode Completion code (output).

pReason Reason code (output).

C message interface

amMsgSetCCSID

Sets the coded character set identifier of the message.

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

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

amMsgSetCorrelId

Sets the correlation identifier of the message.

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

hMsg The message handle returned by **amSesCreateMessage** (input).
correlIdLen The length of the correlation identifier, in bytes (input).
pCorrelId The correlation identifier (input). Specify as NULL (with a **correlIdLen** of 0L) to set the correlation identifier to NULL.
pCompCode Completion code (output).
pReason Reason code (output).

amMsgSetDataOffset

Sets the data offset for reading or writing byte data. If the data offset is greater than the current data length, it is valid to write data into the message at that offset, but an attempt to read data will result in an error. See “**amMsgReadBytes**” on page 103 and “**amMsgWriteBytes**” on page 107.

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

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

amMsgSetElementCCSID

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

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

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

elementCCSID The element coded character set identifier (input).

pCompCode Completion code (output).

pReason Reason code (output).

amMsgSetEncoding

Sets the encoding of the data in the message.

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

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

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

```
AMENC_NATIVE
AMENC_NORMAL
AMENC_NORMAL_FLOAT_390
AMENC_REVERSED
AMENC_REVERSED_FLOAT_390
AMENC_UNDEFINED
```

pCompCode Completion code (output).

pReason Reason code (output).

C message interface

amMsgSetFormat

Sets the format of the message.

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

- hMsg** The message handle returned by **amSesCreateMessage** (input).
- formatLen** The length of the format, in bytes (input). A value of **AMLEN_NULL_TERM** specifies that the string is NULL terminated.
- pFormat** The format of the message (input). It can take one of the following values, or an application defined string:
AMFMT_NONE
AMFMT_STRING
AMFMT_RF_HEADER
- If set to **AMFMT_NONE**, the default format for the sender will be used (if available).
- pCompCode** Completion code (output).
- pReason** Reason code (output).

amMsgSetGroupStatus

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

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

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

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

- hMsg** The message handle returned by **amSesCreateMessage** (input).
- status** The group status (input). It can take one of the following values:
AMGRP_MSG_NOT_IN_GROUP
AMGRP_FIRST_MSG_IN_GROUP
AMGRP_MIDDLE_MSG_IN_GROUP
AMGRP_LAST_MSG_IN_GROUP
AMGRP_ONLY_MSG_IN_GROUP
- pCompCode** Completion code (output).
- pReason** Reason code (output).

amMsgWriteBytes

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

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

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

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

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

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

pByteData The data bytes (input).

pCompCode Completion code (output).

pReason Reason code (output).

Message interface helper macros

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

AmMsgAddStreamName

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

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

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

streamNameLen The length of the stream name, in bytes (input).

pStreamName The stream name (input).

pCompCode Completion code (output).

pReason Reason code (output).

AmMsgGetPubTimeStamp

Gets the publication time stamp name/value element.

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

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

buffLen The length in bytes of a buffer in which the publication time stamp is returned (input). Specify as zero to return only the length.

pTimeStampLen The length of the publication time stamp, in bytes (output). If specified as NULL, the length is not returned.

pTimestamp The publication time stamp (output).

pCompCode Completion code (output).

pReason Reason code (output).

AmMsgGetStreamName

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

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

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

buffLen The length in bytes of a buffer in which the stream name is returned (input). Specify as zero to return only the length.

pStreamNameLen The length of the stream name, in bytes (output). If specified as NULL, the length is not returned.

pStreamName The stream name (output).

pCompCode Completion code (output).

pReason Reason code (output).

Sender interface functions

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

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

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

A *response* sender service is a special type of sender service used for sending a response to a request message. It must be created using the default definition, and not a definition stored in a repository (see “Services and policies” on page 469). Once created, it must not be opened until used in its correct context as a response sender when receiving a request message with **amRcvReceive** or **amReceiveRequest**. When opened, its queue and queue manager properties are modified to reflect the *ReplyTo* destination specified in the message being received. When first used in this context, the sender service becomes a response sender service.

amSndClearErrorCodes

Clears the error codes in the sender object.

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

hSender The sender handle returned by **amSesCreateSender** (input).

pCompCode Completion code (output).

pReason Reason code (output).

amSndClose

Closes the sender service.

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

hSender The sender handle returned by **amSesCreateSender** (input).

hPolicy The handle of a policy (input). If specified as **AMH_NULL_HANDLE**, the system default policy (constant: **AMSD_POL_HANDLE**) is used.

pCompCode Completion code (output).

pReason Reason code (output).

amSndGetCCSID

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

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

hSender The sender handle returned by **amSesCreateSender** (input).

pCCSID The coded character set identifier (output).

pCompCode Completion code (output).

pReason Reason code (output).

amSndGetEncoding

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

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

hSender The sender handle returned by **amSesCreateSender** (input).

pEncoding The encoding (output).

pCompCode Completion code (output).

pReason Reason code (output).

amSndGetLastError

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

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

hSender The sender handle returned by **amSesCreateSender** (input).

buffLen Reserved, must be zero (input).

pStringLen Reserved, must be NULL (input).

pErrorText Reserved, must be NULL (input).

pReason2 A secondary reason code (output). Not returned if specified as NULL. If pReason indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, pReason2 gives 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 <code>AMRC_SERVICE_HANDLE_ERR</code> 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 <code>AMH_NULL_HANDLE</code> , the system default policy (constant: <code>AMSD_POL_HANDLE</code>) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSndSend

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

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

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

hSender	The sender handle returned by amSesCreateSender (input).
hPolicy	The handle of a policy (input). If specified as <code>AMH_NULL_HANDLE</code> , the system default policy (constant: <code>AMSD_POL_HANDLE</code>) 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 <code>AMH_NULL_HANDLE</code> 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 <code>AMH_NULL_HANDLE</code> 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 (<code>hSndMsg</code>).
pData	The message data, if <code>dataLen</code> is non-zero (input).
hSndMsg	The handle of a message object that specifies the properties of the message being sent (input). If <code>dataLen</code> is zero, it can also contain the message data. If specified as <code>AMH_NULL_HANDLE</code> , the default message object (constant: <code>AMSD_SND_MSG_HANDLE</code>) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

C sender interface

amSndSendFile

Sends data from a file. The file data can be received as normal message data by a target application using **amRcvReceive** or used to reconstruct the file with **amRcvReceiveFile**.

```
AMBOOL amSndSendFile(  
    AMHSND    hSender,  
    AMHPOL    hPolicy,  
    AMLONG    options,  
    AMLONG    directoryLen,  
    AMSTR     directory,  
    AMLONG    fileNameLen,  
    AMSTR     fileName,  
    AMHMSG    hSndMsg,  
    PAMLONG    pCompCode,  
    PAMLONG    pReason);
```

Parameters

hSender	The sender handle returned by amSesCreateSender (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE , the system default policy (constant: AMSD_POL_HANDLE) is used.
options	A reserved field that must be specified as zero.
directoryLen	A reserved field that must be specified as zero (input).
directory	A reserved field that must be specified as NULL (input).
fileNameLen	The length of the file name in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
fileName	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the filename will travel with the message for use with a receive file call (see “amRcvReceiveFile” on page 124 for more details). Note that the filename sent will exactly match the supplied filename; it will not be converted or expanded in any way.
hSndMsg	The handle of the message object to use to send the file (input). This can be used to specify the Correlation ID for example. If specified as AMH_NULL_HANDLE , the system default send message (constant: AMSD_SND_MSG_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

If, in your application, you have previously used a message object, referenced by either handle or name, to send or receive data (including AMI elements or topics), you will need to explicitly call **amMsgReset** before re-using the object for sending a file. This applies even if you use the system default object handle (constant: **AMSD_SND_MSG_HANDLE**).

Receiver interface functions

A *receiver* object encapsulates an MQSeries object descriptor (MQOD) structure. This represents a local MQSeries queue. An open receiver service is always associated with an open connection object, such as a queue manager connection. Support is also included for dynamic receiver services (that encapsulate model queues). The required receiver service object definitions can be provided from a repository or can be created automatically from the set of existing queue objects available on the local queue manager.

There is a definition type associated with each receiver service:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

A receiver service created from a repository definition will be initially of type `AMDT_PREDEFINED` or `AMDT_DYNAMIC`. When opened, its definition type might change from `AMDT_DYNAMIC` to `AMDT_TEMP_DYNAMIC` according to the properties of its underlying queue object.

A receiver service created with default values (that is, without a repository definition) will have its definition type set to `AMDT_UNDEFINED` until it is opened. When opened, this will become `AMDT_DYNAMIC`, `AMDT_TEMP_DYNAMIC`, or `AMDT_PREDEFINED`, according to the properties of its underlying queue object.

amRcvBrowse

Browses a message. See the *MQSeries Application Programming Guide* for a full description of the browse options.

```
AMBOOL amRcvBrowse(
    AMHRCV    hReceiver,
    AMHPOL    hPolicy,
    AMLONG    options,
    AMLONG    buffLen,
    PAMLONG   pDataLen,
    PAMBYTE   pData,
    AMHMSG    hRcvMsg,
    AMHSND    hSender,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

hReceiver The receiver handle returned by `amSesCreateReceiver` (input).

hPolicy The handle of a policy (input). If specified as `AMH_NULL_HANDLE`, the system default policy (constant: `AMSD_POL_HANDLE`) is used.

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

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

C receiver interface

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

Note that a locked message is unlocked by another browse or receive, even though it is not for the same message. The locking feature is not available on OS/390.

buffLen	The length in bytes of a buffer in which the data is returned (input).
pDataLen	The length of the message data in bytes (output). If specified as NULL, the data length is not returned.
pData	The received message data (output).
hRcvMsg	The handle of the message object for the received message (output).
hSender	The handle of the response sender service that the response message must be sent to, if this is a request message (output). This sender service must be created without a repository definition, and used exclusively for sending a response. Its definition type must be AMDT_UNDEFINED (it will be set to AMDT_RESPONSE by this call).
	Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

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

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

To return data in an application message buffer:

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

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

To return only the data length:

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

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

amRcvBrowseSelect

Browses a message identified by specifying the Correlation ID from the selection message as a selection criterion. See the *MQSeries Application Programming Guide* for a full description of the browse options.

```
AMBOOL amRcvBrowseSelect(
    AMHRCV    hReceiver,
    AMHPOL    hPolicy,
    AMLONG    options,
    AMHMSG    hSelMsg,
    AMLONG    buffLen,
    PAMLONG   pDataLen,
    PAMBYTE   pData,
    AMHMSG    hRcvMsg,
    AMHSND    hSender,
    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.

hSelMsg The handle of a selection message object (input). This is used together with the browse options to identify the message to be received (for example, using the Correlation ID). Specify as **AMH_NULL_HANDLE** to get the next available message. The CCSID, element CCSID, and encoding values from the selection message define the target values for any data conversions. If target conversion values are required without using the Correlation ID for selection, this can be reset (see **amMsgSetCorrelId** on page 104) before invoking the **amRcvBrowseSelect** function.

buffLen The length in bytes of a buffer in which the data is returned (input).

pDataLen The length of the message data in bytes (output). If specified as **NULL**, the data length is not returned.

pData The received message data (output).

hRcvMsg The handle of the message object for the received message (output).

C receiver interface

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

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

pCompCode Completion code (output).

pReason Reason code (output).

Usage notes

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

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

To return data in an application message buffer:

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

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

To return only the data length:

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

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

amRcvClearErrorCodes

Clears the error codes in the receiver service object.

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

hReceiver The receiver handle returned by `amSesCreateReceiver` (input).

pCompCode Completion code (output).

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).

pReason Reason code (output).

C receiver interface

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).
pReason	Reason code (output).

amRcvGetQueueName

Gets the queue name of the receiver service. This is used to determine the queue name of a permanent dynamic receiver service, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **amRcvSetQueueName**.)

```
AMBOOL amRcvGetQueueName(
    AMHRCV    hReceiver,
    AMLONG    buffLen,
    PAMLONG   pNameLen,
    AMSTR     pQueueName,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

hReceiver	The receiver handle returned by amSesCreateReceiver (input).
buffLen	The length in bytes of a buffer in which the queue name is returned (input).
pNameLen	The length of the queue name, in bytes (output).
pQueueName	The queue name of the receiver service (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amRcvOpen

Opens the receiver service.

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

hReceiver	The receiver handle returned by amSesCreateReceiver (input).
	The handle of a policy (input). If specified as AMH_NULL_HANDLE , the system default policy (constant: AMSD_POL_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

C receiver interface

amRcvReceive

Receives a message.

```
AMBOOL amRcvReceive(  
    AMHRCV    hReceiver,  
    AMHPOL    hPolicy,  
    AMHMSG    hSelMsg,  
    AMLONG    buffLen,  
    PAMLONG    pDataLen,  
    PAMBYTE    pData,  
    AMHMSG    hRcvMsg,  
    AMHSND    hSender,  
    PAMLONG    pCompCode,  
    PAMLONG    pReason);
```

hReceiver	The receiver handle returned by amSesCreateReceiver (input).
hPolicy	The handle of a policy (input). If specified as <code>AMH_NULL_HANDLE</code> , the system default policy (constant: <code>AMSD_POL_HANDLE</code>) 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 <code>AMH_NULL_HANDLE</code> to get the next available message with no selection. The <code>CCSID</code> , element <code>CCSID</code> , and encoding values from the selection message define the target values for any data conversions. If target conversion values are required without using the Correlation ID for selection, this can be reset (see amMsgSetCorrelId on page 78) before invoking the amRcvReceive function.
buffLen	The length in bytes of a buffer in which the data is returned (input).
pDataLen	The length of the message data, in bytes (output). If specified as <code>NULL</code> , 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 <code>AMH_NULL_HANDLE</code> , the default message object (constant: <code>AMSD_RCV_MSG_HANDLE</code>) 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 <code>AMDT_UNDEFINED</code> (it will be set to <code>AMDT_RESPONSE</code> by this call).
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

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

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

To return data in an application message buffer:

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

- set `buffLen` to the length of the buffer

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

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

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

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

To remove the message from the queue and discard it:

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

The message will be discarded with an `AMRC_MSG_TRUNCATED` warning.

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

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

C receiver interface

amRcvReceiveFile

Receives file message data into a file.

```
AMBOOL amRcvReceiveFile(  
    AMHRCV    hReceiver,  
    AMHPOL    hPolicy,  
    AMHLONG   options,  
    AMHMSG    hSelMsg,  
    AMLONG    directoryLen,  
    AMSTR     directory,  
    AMLONG    fileNameLen,  
    AMSTR     fileName,  
    AMHMSG    hRcvMsg,  
    PAMLONG   pCompCode,  
    PAMLONG   pReason);
```

hReceiver	The receiver handle returned by amSesCreateReceiver (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE , the system default policy (constant: AMSD_POL_HANDLE) is used.
options	A reserved field that must be specified as zero (input).
hSelMsg	The handle of a selection message object (input). This is used to identify the message to be received (for example, using the correlation ID). Specify as AMH_NULL_HANDLE to get the next available message with no selection. The CCSID , element CCSID , and encoding values from the selection message define the target values for any data conversions. If target conversion values are required without using the Correlation ID for selection, this can be reset (see amMsgSetCorrelId on page 104) before invoking the amRcvReceiveFile function.
directoryLen	A reserved field that must be specified as zero (input).
directory	A reserved field that must be specified as NULL (input).
fileNameLen	The length of the file name in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated, in which case the AMI will work out the length itself.
fileName	The name of the file into which the transferred data is to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If NULL or a null string is specified, the AMI will use the name of the originating file (including any directory prefix), exactly as it was supplied on the send file call. Note that the original filename may not be appropriate for use by the receiver, either because a pathname included in the filename is not applicable to the receiving system, or because the sending and receiving systems use different filename conventions.
hRcvMsg	The handle of the message object to use to receive the file. This parameter is updated with the message properties, for example the Message ID. If the message is not a file message, hRcvMsg receives the message data. If hRcvMsg is specified as AMH_NULL_HANDLE , the default message object (constant AMSD_RCV_MSG_HANDLE) is used. The message object is reset implicitly before the receive takes place.
pCompCode	Completion code (output).

pReason Reason code (output).

Usage notes

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

amRcvSetQueueName

Sets the queue name of the receiver service, when this encapsulates a model queue. This can be used to specify the queue name of a recreated permanent dynamic receiver service, in order to receive messages in a session subsequent to the one in which it was created. (See also **amRcvGetQueueName**.)

```
AMBOOL amRcvSetQueueName(
    AMHRCV    hReceiver,
    AMLONG    nameLen,
    AMSTR     pQueueName,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

hReceiver The receiver handle returned by **amSesCreateReceiver** (input).

nameLen The length of the queue name, in bytes (input). A value of `AMLEN_NULL_TERM` specifies that the string is NULL terminated.

pQueueName The queue name of the receiver service (input).

pCompCode Completion code (output).

pReason Reason code (output).

Distribution list interface functions

A *distribution list* object encapsulates a list of sender objects.

amDstClearErrorCodes

Clears the error codes in the distribution list object.

```
AMBOOL amDstClearErrorCodes(  
    AMHDST    hDistList,  
    PAMLONG   pCompCode,  
    PAMLONG   pReason);
```

hDistList The distribution list handle returned by **amSesCreateDistList** (input).

pCompCode Completion code (output).

pReason Reason code (output).

amDstClose

Closes the distribution list.

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

hDistList The distribution list handle returned by **amSesCreateDistList** (input).

hPolicy The handle of a policy (input). If specified as **AMH_NULL_HANDLE**, the system default policy (constant: **AMSD_POL_HANDLE**) is used.

pCompCode Completion code (output).

pReason Reason code (output).

amDstGetLastError

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

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

hDistList	The distribution list handle returned by amSesCreateDistList (input).
buffLen	Reserved, must be zero (input).
pStringLen	Reserved, must be NULL (input).
pErrorText	Reserved, must be NULL (input).
pReason2	A secondary reason code (output). Not returned if specified as NULL. If pReason indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, pReason2 gives an MQSeries reason code.
pCompCode	Completion code (output). Not returned if specified as NULL.
pReason	Reason code (output). Not returned if specified as NULL. A value of AMRC_SERVICE_HANDLE_ERR indicates that the amDstGetLastError function call has itself detected an error and failed.

amDstGetName

Gets the name of the distribution list object.

```
AMBOOL amDstGetName(
    AMHDST hDistList,
    AMLONG buffLen,
    PAMLONG pNameLen,
    AMSTR pName,
    PAMLONG pCompCode,
    PAMLONG pReason);
```

hDistList	The distribution list handle returned by amSesCreateDistList (input).
buffLen	The length in bytes of a buffer into which the name is put (input). Set it to zero to return only the name length.
pNameLen	The length of the name, in bytes (output). Set it to NULL to return only the name.
pName	The distribution list object name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

C distribution list interface

amDstGetSenderCount

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

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

hDistList The distribution list handle returned by **amSesCreateDistList** (input).

pCount The number of sender services (output).

pCompCode Completion code (output).

pReason Reason code (output).

amDstGetSenderHandle

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

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

hDistList The distribution list handle returned by **amSesCreateDistList** (input).

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

pCompCode Completion code (output).

pReason Reason code (output).

amDstOpen

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

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

hDistList The distribution list handle returned by **amSesCreateDistList** (input).

hPolicy The handle of a policy (input). If specified as **AMH_NULL_HANDLE**, the system default policy (constant: **AMSD_POL_HANDLE**) is used.

pCompCode Completion code (output).

pReason Reason code (output).

amDstSend

Sends a message to each sender in the distribution list.

```
AMBOOL amDstSend(
    AMHDST    hDistList,
    AMHPOL    hPolicy,
    AMHRCV    hReceiver,
    AMLONG    dataLen,
    PAMBYTE   pData,
    AMHMSG    hMsg,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

hDistList	The distribution list handle returned by amSesCreateDistList (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE , the system default policy (constant: AMSD_POL_HANDLE) is used.
hReceiver	The handle of the receiver service to which the response to this message should be sent, if the message being sent is a request message (input). Specify as AMH_NULL_HANDLE if no response is required.
dataLen	The length of the message data, in bytes (input). If set to zero, the data should be passed in the message object (hMsg).
pData	The message data (input).
hMsg	The handle of a message object that specifies the properties for the message being sent (input). If dataLen is zero, it should also contain the message data. If specified as AMH_NULL_HANDLE , the default send message object (constant: AMSD_SND_MSG_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

C distribution list interface

amDstSendFile

Sends data from a file to each sender in the distribution list. The file data can be received as normal message data by a target application using **amRcvReceive** or used to reconstruct the file with **amRcvReceiveFile**.

```
AMBOOL amDstSendFile(  
    AMHDST    hDistList,  
    AMHPOL    hPolicy,  
    AMLONG    options,  
    AMLONG    directoryLen,  
    AMSTR     directory,  
    AMLONG    fileNameLen,  
    AMSTR     fileName,  
    AMHMSG    hMsg,  
    PAMLONG   pCompCode,  
    PAMLONG   pReason);
```

Parameters

hDistList	The distribution list handle returned by amSesCreateDistList (input).
hPolicy	The handle of a policy (input). If specified as AMH_NULL_HANDLE , the system default policy (constant: AMSD_POL_HANDLE) is used.
options	Reserved, must be specified as 0L (input).
directoryLen	A reserved field that must be specified as zero (input).
directory	A reserved field that must be specified as NULL (input).
fileNameLen	The length of the file name in bytes (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
fileName	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the filename will travel with the message for use with a receive file call (see “ amRcvReceiveFile ” on page 124 for more details). Note that the filename sent will exactly match the supplied filename; it will not be converted or expanded in any way.
hMsg	The handle of the message object to use to send the file (input). This can be used to specify the Correlation ID for example. If specified as ANM_NULL_HANDLE , the default send message object (constant: AMSD_SND_MSG_HANDLE) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Usage notes

If, in your application, you have previously used a message object, referenced by either handle or name, to send or receive data (including AMI elements or topics), you will need to explicitly call **amMsgReset** before re-using the object for sending a file. This applies even if you use the system default object handle (constant: **AMSD_SND_MSG_HANDLE**).

The system default message object handle is used when you specify **hMsg** as **AMH_NULL_HANDLE**.

Publisher interface functions

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

amPubClearErrorCodes

Clears the error codes in the publisher object.

```
AMBOOL amPubClearErrorCodes(
    AMHPUB    hPublisher,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

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

pCompCode Completion code (output).

pReason Reason code (output).

amPubClose

Closes the publisher service.

```
AMBOOL amPubClose(
    AMHPUB    hPublisher,
    AMHPOL    hPolicy,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

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

hPolicy The handle of a policy (input). If specified as **AMH_NULL_HANDLE**, the system default policy (constant: **AMSD_POL_HANDLE**) is used.

pCompCode Completion code (output).

pReason Reason code (output).

amPubGetCCSID

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

```
AMBOOL amPubGetCCSID(
    AMHPUB    hPublisher,
    PAMLONG   pCCSID,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

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

pCCSID The coded character set identifier (output).

pCompCode Completion code (output).

pReason Reason code (output).

C publisher interface

amPubGetEncoding

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

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

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

pEncoding The encoding (output).

pCompCode Completion code (output).

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 **amPubGetLastError** function call has itself detected an error and failed.

amPubGetName

Gets the name of the publisher service.

```

AMBOOL amPubGetName(
    AMHPUB    hPublisher,
    AMLONG    buffLen,
    PAMLONG   pNameLen,
    AMSTR     pName,
    PAMLONG   pCompCode,
    PAMLONG   pReason);

```

hPublisher	The publisher handle returned by amSesCreatePublisher (input).
buffLen	The length in bytes of a buffer into which the name is put (input). Set it to zero to return only the name length.
pNameLen	The length of the name, in bytes (output). Set it to NULL to return only the name.
pName	The publisher object name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amPubOpen

Opens the publisher service.

```

AMBOOL amPubOpen(
    AMHPUB    hPublisher,
    AMHPOL    hPolicy,
    PAMLONG   pCompCode,
    PAMLONG   pReason);

```

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

C publisher interface

amPubPublish

Publishes a message using the publisher service.

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

```
AMBOOL amPubPublish(  
    AMHPUB    hPublisher,  
    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 <code>AMH_NULL_HANDLE</code> , the system default policy (constant: <code>AMSD_POL_HANDLE</code>) is used.
hReceiver	The handle of the receiver service to which the response to this publish request should be sent (input). Specify as <code>AMH_NULL_HANDLE</code> 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 <code>AMH_NULL_HANDLE</code> , the default message object (constant: <code>AMSD_SND_MSG_HANDLE</code>) is used.
pCompCode	Completion code (output).
pReason	Reason code (output).

Subscriber interface functions

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

amSubClearErrorCodes

Clears the error codes in the subscriber object.

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

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

pCompCode Completion code (output).

pReason Reason code (output).

amSubClose

Closes the subscriber service.

```
AMBOOL amSubClose(
    AMHSUB    hSubscriber,
    AMHPOL    hPolicy,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

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

hPolicy The handle of a policy (input). If specified as **AMH_NULL_HANDLE**, the system default policy (constant: **AMSD_POL_HANDLE**) is used.

pCompCode Completion code (output).

pReason Reason code (output).

amSubGetCCSID

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

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

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

pCCSID The coded character set identifier (output).

pCompCode Completion code (output).

pReason Reason code (output).

C subscriber interface

amSubGetDefnType

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

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

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

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

```
AMDT_UNDEFINED  
AMDT_TEMP_DYNAMIC  
AMDT_DYNAMIC  
AMDT_PREDEFINED
```

pCompCode Completion code (output).

pReason Reason code (output).

amSubGetEncoding

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

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

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

pEncoding The encoding (output).

pCompCode Completion code (output).

pReason Reason code (output).

amSubGetLastError

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

```
AMBOOL amSubGetLastError(
    AMHSUB    hSubscriber,
    AMLONG    buffLen,
    PAMLONG   pStringLen,
    AMSTR     pErrorText,
    PAMLONG   pReason2,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

- hSubscriber** The subscriber handle returned by **amSesCreateSubscriber** (input).
- buffLen** Reserved, must be zero (input).
- pStringLen** Reserved, must be NULL (input).
- pErrorText** Reserved, must be NULL (input).
- pReason2** A secondary reason code (output). Not returned if specified as NULL. If pReason indicates AMRC_TRANSPORT_WARNING or AMRC_TRANSPORT_ERR, pReason2 gives an MQSeries reason code.
- pCompCode** Completion code (output). Not returned if specified as NULL.
- pReason** Reason code (output). Not returned if specified as NULL. A value of AMRC_SERVICE_HANDLE_ERR indicates that the **amSubGetLastError** function call has itself detected an error and failed.

amSubGetName

Gets the name of the subscriber object.

```
AMBOOL amSubGetName(
    AMHSUB    hSubscriber,
    AMLONG    buffLen,
    PAMLONG   pNameLen,
    AMSTR     pName,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

- hSubscriber** The subscriber handle returned by **amSesCreateSubscriber** (input).
- buffLen** The length in bytes of a buffer into which the name is put (input). Set it to zero to return only the name length.
- pNameLen** The length of the name, in bytes (output). Set it to NULL to return only the name.
- pName** The subscriber object name (output).
- pCompCode** Completion code (output).
- pReason** Reason code (output).

C subscriber interface

amSubGetQueueName

Gets the queue name of the subscriber's receiver service object. This can be used to determine the queue name of a permanent dynamic receiver service, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **amSubSetQueueName**.)

```
AMBOOL amSubGetQueueName(  
    AMHSUB    hSubscriber,  
    AMLONG    buffLen,  
    PAMLONG    pStringLen,  
    AMSTR     pQueueName,  
    PAMLONG    pCompCode,  
    PAMLONG    pReason);
```

- hSubscriber** The subscriber handle returned by **amSesCreateSubscriber** (input).
- buffLen** The length in bytes of a buffer in which the queue name is returned (input). Specify as zero to return only the length.
- pStringLen** The length of the queue name, in bytes (output). If specified as NULL, the length is not returned.
- pQueueName** The queue name (output).
- pCompCode** Completion code (output).
- pReason** Reason code (output).

amSubOpen

Opens the subscriber service.

```
AMBOOL amSubOpen(  
    AMHSUB    hSubscriber,  
    AMHPOL    hPolicy,  
    PAMLONG    pCompCode,  
    PAMLONG    pReason);
```

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

amSubReceive

Receives a message, normally a publication, using the subscriber service. The message data, topic and other elements can be accessed using the message interface functions (see page 90).

The message data is passed in the message object. There is no option to pass it as a separate parameter as with **amRcvReceive** (this would not give any performance improvement because the MQRFH header has to be removed from the message data after receiving it).

```
AMBOOL amSubReceive(
    AMHSUB    hSubscriber,
    AMHPOL    hPolicy,
    AMHMSG    hSelMsg,
    AMHMSG    hRcvMsg,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by amSesCreateSubscriber (input).
hPolicy	The handle of a policy (input). If specified as <code>AMH_NULL_HANDLE</code> , the system default policy (constant: <code>AMSD_POL_HANDLE</code>) 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 <code>AMH_NULL_HANDLE</code> to get the next available message with no selection.
hRcvMsg	The handle of the message object for the received message (output). If specified as <code>AMH_NULL_HANDLE</code> , the default message object (constant: <code>AMSD_RCV_MSG_HANDLE</code>) is used. The message object is reset implicitly before the receive takes place.
pCompCode	Completion code (output).
pReason	Reason code (output).

amSubSetQueueName

Sets the queue name of the subscriber's receiver object, when this encapsulates a model queue. This can be used to specify the queue name of a recreated permanent dynamic receiver service, in order to receive messages in a session subsequent to the one in which it was created. (See also **amSubGetQueueName**.)

```
AMBOOL amSubSetQueueName(
    AMHSUB    hSubscriber,
    AMLONG    nameLen,
    AMSTR     pQueueName,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

hSubscriber	The subscriber handle returned by amSesCreateSubscriber (input).
nameLen	The length of the queue name, in bytes (input).
pQueueName	The queue name (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

C subscriber interface

amSubSubscribe

Sends a subscribe message to a publish/subscribe broker using the subscriber service, to register a subscription. The topic and other elements can be specified using the message interface functions (see page 90) before sending the message.

Publications matching the subscription are sent to the receiver service associated with the subscriber. By default, this has the same name as the subscriber service, with the addition of the suffix '.RECEIVER'.

```
AMBOOL amSubSubscribe(  
    AMHSUB    hSubscriber,  
    AMHPOL    hPolicy,  
    AMHRCV    hReceiver,  
    AMHMSG    hSubMsg,  
    PAMLONG   pCompCode,  
    PAMLONG   pReason);
```

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

hPolicy The handle of a policy (input). If specified as **AMH_NULL_HANDLE**, the system default policy (constant: **AMSD_POL_HANDLE**) is used.

hReceiver The handle of the receiver service to which the response to this subscribe request should be sent (input). Specify as **AMH_NULL_HANDLE** if no response is required.

This is not the service to which publications will be sent by the broker; they are sent to the receiver service associated with the subscriber (see above).

hSubMsg The handle of a message object for the subscribe message (input). If specified as **AMH_NULL_HANDLE**, the default message object (constant: **AMSD_SND_MSG_HANDLE**) is used.

pCompCode Completion code (output).

pReason Reason code (output).

amSubUnsubscribe

Sends an unsubscribe message to a publish/subscribe broker using the subscriber service, to deregister a subscription. The topic and other elements can be specified using the message interface functions (see page 90) before sending the message.

To deregister all topics, a policy providing this option must be specified (this is not the default policy). Otherwise, to remove a previous subscription the topic information specified must match that specified on the relevant **amSubSubscribe** request.

```
AMBOOL amSubUnsubscribe(
    AMHSUB    hSubscriber,
    AMHPOL    hPolicy,
    AMHRCV    hReceiver,
    AMHMSG    hUnsubMsg,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

- hSubscriber** The subscriber handle returned by **amSesCreateSubscriber** (input).
- hPolicy** The handle of a policy (input). If specified as **AMH_NULL_HANDLE**, the system default policy (constant: **AMSD_POL_HANDLE**) is used.
- hReceiver** The handle of the receiver service to which the response to this unsubscribe request should be sent (input). Specify as **AMH_NULL_HANDLE** if no response is required.
- hUnsubMsg** The handle of a message object for the unsubscribe message (input). If specified as **AMH_NULL_HANDLE**, the default message object (constant: **AMSD_SND_MSG_HANDLE**) is used.
- pCompCode** Completion code (output).
- pReason** Reason code (output).

Policy interface functions

A *policy* object encapsulates the set of options used for each AMI request (open, close, send, receive, publish and so on). Examples are the priority and persistence of the message, and whether the message is included in a unit of work.

amPolClearErrorCodes

Clears the error codes in the policy object.

```
AMBOOL amPolClearErrorCodes(  
    AMHPOL    hPolicy,  
    PAMLONG   pCompCode,  
    PAMLONG   pReason);
```

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

pCompCode Completion code (output).

pReason Reason code (output).

amPolGetLastError

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

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

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

buffLen Reserved, must be zero (input).

pStringLen Reserved, must be NULL (input).

pErrorText Reserved, must be NULL (input).

pReason2 A secondary reason code (output). Not returned if specified as NULL. If **pReason** indicates **AMRC_TRANSPORT_WARNING** or **AMRC_TRANSPORT_ERR**, **pReason2** gives an MQSeries reason code.

pCompCode Completion code (output). Not returned if specified as NULL.

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

amPolGetName

Returns the name of the policy object.

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

hPolicy	The policy handle returned by amSesCreatePolicy (input).
buffLen	The length in bytes of a buffer into which the name is put (input). Set it to zero to return only the name length.
pNameLen	The length of the name, in bytes (output). Set it to NULL to return only the name.
pName	The policy object name (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amPolGetWaitTime

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

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

hPolicy	The policy handle returned by amSesCreatePolicy (input).
pWaitTime	The wait time, in ms (output).
pCompCode	Completion code (output).
pReason	Reason code (output).

amPolSetWaitTime

Sets the wait time for any receive function using this policy.

```
AMBOOL amPolSetWaitTime(
    AMHPOL    hPolicy,
    AMLONG    waitTime,
    PAMLONG   pCompCode,
    PAMLONG   pReason);
```

hPolicy	The policy handle returned by amSesCreatePolicy (input).
waitTime	The wait time (in ms) to be set in the policy (input).
pCompCode	Completion code (output).
pReason	Reason code (output).

C policy interface

Part 3. The C++ interface

Chapter 6. Using the Application Messaging

Interface in C++	149
Structure of the AMI	149
Base classes	149
Interface and helper classes.	150
Exception classes	150
Using the repository	150
System default objects	150
Writing applications in C++	151
Creating and opening objects	151
Deleting objects	152
Sending messages	152
Sample program	153
Receiving messages	153
Sample program	154
Request/response messaging	154
Sample programs	155
File transfer	155
Publish/subscribe messaging	156
Sample programs	157
Using AmElement objects	157
Error handling	157
Transaction support	158
Sending group messages	159
Other considerations	159
Multithreading	159
Using MQSeries with the AMI.	159
Field limits	160
Building C++ applications	161
AMI include files	161
C++ applications on AIX	161
Preparing C++ programs on AIX	161
Running C++ programs on AIX	162
C++ applications on AS/400	162
Preparing C++ programs on AS/400.	162
Running C++ programs on AS/400	163
C++ applications on HP-UX	163
Preparing C++ programs on HP-UX.	163
Running C++ programs on HP-UX	163
C++ applications on Solaris.	164
Preparing C++ programs on Solaris	164
Running C++ programs on Solaris	164
C++ applications on Windows.	165
Preparing C++ programs on Windows	165
Running C++ programs on Windows	165

Chapter 7. C++ interface overview 167

Base classes	167
Helper classes	167
Exception classes	167
AmSessionFactory	168
Constructor	168
Session factory management	168
Create and delete session	168
AmSession	169
Session management	169

Create objects	169
Delete objects	169
Transactional processing.	169
Error handling	170
AmMessage	171
Get values	171
Set values	171
Reset values	171
Read and write data	171
Publish/subscribe topics.	172
Publish/subscribe filters.	172
Publish/subscribe name/value elements	172
Error handling	172
AmSender	173
Open and close.	173
Send	173
Send file	173
Get values	173
Error handling	173
AmReceiver	174
Open and close.	174
Receive and browse	174
Receive file	174
Get values	174
Set value	174
Error handling	174
AmDistributionList	175
Open and close.	175
Send	175
Send file	175
Get values	175
Error handling	175
AmPublisher	176
Open and close.	176
Publish	176
Get values	176
Error handling	176
AmSubscriber	177
Open and close.	177
Broker messages	177
Get values	177
Set value	177
Error handling	177
AmPolicy.	178
Policy management	178
Error handling	178
Helper classes	179
AmBytes	179
AmElement	179
AmObject	179
AmStatus.	179
AmString.	180
Exception classes	181
AmException	181
AmErrorException.	181
AmWarningException	181

Chapter 8. C++ interface reference	183		
Base classes	183	getFilter	193
Helper classes	183	getFilterCount	193
Exception classes	183	getFormat	193
AmSessionFactory	184	getGroupStatus	194
AmSessionFactory	184	getLastErrorStatus	194
createSession	184	getMessageId	194
deleteSession	184	getName	194
getFactoryName	184	getNamedElement	194
getLocalHost	184	getNamedElementCount	194
getRepository	184	getReportCode	195
getTraceLevel	184	getTopic	195
getTraceLocation	184	getTopicCount	195
setLocalHost	185	getType	195
setRepository	185	readBytes	195
setTraceLevel	185	reset	195
setTraceLocation	185	setCCSID	196
AmSession	186	setCorrelationId	196
begin	186	setDataOffset	196
clearErrorCodes	186	setElementCCSID	196
close	186	setEncoding	196
commit	186	setFormat	197
createDistributionList	186	setGroupStatus	197
createMessage	187	writeBytes	197
createPolicy	187	AmSender	198
createPublisher	187	clearErrorCodes	198
createReceiver	187	close	198
createSender	187	enableWarnings	198
createSubscriber	187	getCCSID	198
deleteDistributionList	188	getEncoding	198
deleteMessage	188	getLastErrorStatus	199
deletePolicy	188	getName	199
deletePublisher	188	open	199
deleteReceiver	188	send	199
deleteSender	188	sendFile	200
deleteSubscriber	188	AmReceiver	201
enableWarnings	188	browse	201
getLastErrorStatus	189	clearErrorCodes	202
getName	189	close	202
getTraceLevel	189	enableWarnings	202
getTraceLocation	189	getDefinitionType	202
open	189	getLastErrorStatus	203
rollback	189	getName	203
AmMessage	190	getQueueName	203
addElement	191	open	203
addFilter	191	receive	203
addTopic	191	receiveFile	204
clearErrorCodes	191	setQueueName	204
deleteElement	191	AmDistributionList	205
deleteFilter	191	clearErrorCodes	205
deleteNamedElement	192	close	205
deleteTopic	192	enableWarnings	205
enableWarnings	192	getLastErrorStatus	205
getCCSID	192	getName	205
getCorrelationId	192	getSender	205
getDataLength	192	getSenderCount	205
getDataOffset	192	open	205
getElement	193	send	206
getElementCCSID	193	sendFile	206
getElementCount	193	AmPublisher	207
getEncoding	193	clearErrorCodes	207
		close	207

enableWarnings	207	destructor	218
getCCSID	207	operators	219
getEncoding	207	pad	219
getLastErrorStatus	207	split	219
getName	207	strip	219
open	208	length	219
publish	208	text	219
AmSubscriber	209	truncate	219
clearErrorCodes	209	AmException	220
close	209	getClassName	220
enableWarnings	209	getCompletionCode	220
getCCSID	209	getMethodName	220
getDefinitionType	209	getReasonCode	220
getEncoding	209	getSource	220
getLastErrorStatus	209	toString	220
getName	210	AmErrorException	221
getQueueName	210	getClassName	221
open	210	getCompletionCode	221
receive	210	getMethodName	221
setQueueName	210	getReasonCode	221
subscribe	211	getSource	221
unsubscribe	211	toString	221
AmPolicy	212	AmWarningException	222
clearErrorCodes	212	getClassName	222
enableWarnings	212	getCompletionCode	222
getLastErrorStatus	212	getMethodName	222
getName	212	getReasonCode	222
getWaitTime	212	getSource	222
setWaitTime	212	toString	222
AmBytes	213		
cmp	213		
constructors	213		
cpy	214		
dataPtr	214		
destructor	214		
length	214		
operators	214		
pad	214		
AmElement	215		
AmElement	215		
getName	215		
getValue	215		
getVersion	215		
setVersion	215		
toString	215		
AmObject	216		
clearErrorCodes	216		
getLastErrorStatus	216		
getName	216		
AmStatus	217		
AmStatus	217		
getCompletionCode	217		
getReasonCode	217		
getReasonCode2	217		
toString	217		
AmString	218		
cat	218		
cmp	218		
constructors	218		
contains	218		
cpy	218		

Chapter 6. Using the Application Messaging Interface in C++

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

This chapter describes the following:

- “Structure of the AMI”
- “Writing applications in C++” on page 151
- “Building C++ applications” on page 161

Note that the term *object* is used in this book in the object-oriented programming sense, not in the sense of MQSeries ‘objects’ such as channels and queues.

Structure of the AMI

The following classes are provided:

Base classes

AmSessionFactory	Creates AmSession objects.
AmSession	Creates objects within the AMI session, and controls transactional support.
AmMessage	Contains the message data, message ID and correlation ID, and options that are used when sending or receiving a message (most of which come from the policy definition).
AmSender	This is a service that represents a destination (such as an MQSeries queue) to which messages are sent.
AmReceiver	This is a service that represents a source (such as an MQSeries queue) from which messages are received.
AmDistributionList	Contains a list of sender services to provide a list of destinations.
AmPublisher	Contains a sender service where the destination is a publish/subscribe broker.
AmSubscriber	Contains a sender service (to send subscribe and unsubscribe messages to a publish/subscribe broker) and a receiver service (to receive publications from the broker).
AmPolicy	Defines how the message should be handled, including items such as priority, persistence, and whether it is included in a unit of work.

Structure of the AMI

Interface and helper classes

AmObject	This is an abstract class, from which the base classes listed previously inherit (with the exception of AmSessionFactory).
AmElement	This encapsulates name/value pairs for use in publish/subscribe applications.
AmStatus	This encapsulates the error status of amCcpp objects.
AmString	This encapsulates string data.
AmBytes	This encapsulates binary/byte data.

Exception classes

AmException	This is the base Exception class for amCcpp; all other amCcpp Exceptions inherit from this class.
AmErrorException	An Exception of this type is raised when an amCcpp object experiences an error with a severity level of FAILED (CompletionCode = AMCC_FAILED).
AmWarningException	An Exception of this type is raised when an amCcpp object experiences an error with a severity level of WARNING (CompletionCode = AMCC_WARNING), provided that warnings have been enabled using the enableWarnings method.

Using the repository

You can run AMI applications with or without a repository. If you do not have a repository, you can create an object by specifying its name in a method. It will be created using the appropriate system provided definition (see "System provided definitions" on page 470).

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

System default objects

The set of system default objects created in C is not accessible directly in C++, but the SYSTEM.DEFAULT.POLICY (constant: AMSD_POL) is used to provide default behavior when a policy is not specified. Objects with identical properties to the system default objects can be created for use in C++ using the built-in definitions (see "System provided definitions" on page 470).

Writing applications in C++

This section gives a number of examples showing how to access the Application Messaging Interface using C++.

Many of the method calls are overloaded and in some cases this results in default objects being used. One example of this is the AmPolicy object which can be passed on many of the methods. For example:

Method overloading

```
mySender->send(*mySendMessage, *myPolicy);

mySender->send(*mySendMessage);
```

If a policy has been created to provide specific send behavior, use the first example. However, if the default policy is acceptable, use the second example.

The defaulting of behavior using method overloading is used throughout the examples.

Creating and opening objects

Before using the AMI, you must create and open the required objects. Objects are created with names, which might correspond to named objects in the repository. In the case of the creation of a response sender (myResponder) in the following example, the default name for a response type object is specified, so the object is created with default responder values.

Creating AMI objects

```
mySessionFactory = new AmSessionFactory("MY.REPOSITORY.XML");
mySession = mySessionFactory->createSession("MY.SESSION");
myPolicy = mySession->createPolicy("MY.POLICY");

mySender = mySession->createSender("AMT.SENDER.QUEUE");
myReceiver = mySession->createReceiver("AMT.RECEIVER.QUEUE");
myResponder = mySession->createSender(AMDEF_RSP_SND);

mySendMessage = mySession->createMessage("MY.SEND.MESSAGE");
myReceiveMessage = mySession->createMessage("MY.RECEIVE.MESSAGE");
```

The objects are then opened. In the following examples, the session object is opened with the default policy, whereas the sender and receiver objects are opened with a specified policy (myPolicy).

Opening the AMI objects

```
mySession->open();
mySender->open(*myPolicy);
myReceiver->open(*myPolicy);
```


Writing applications in C++

Deleting objects

To avoid memory leaks, it is essential to explicitly delete all C++ objects that you have created at the end of your program. Delete the session after everything other than the session factory. Delete the session factory last.

The following is an example from the `Receiver.cpp` sample program:

Deleting AMI objects

```
mySession->deleteMessage(myReceiveMsg);
mySession->deleteReceiver(myReceiver);
mySession->deletePolicy(myPolicy);
mySessionFactory->deleteSession(mySession);
delete *mySessionFactory;
```

Sending messages

The examples in this section show how to send a datagram (send and forget) message. First, the message data is written to the `mySendMessage` object. Data is always sent in byte form using the `AmBytes` helper class.

Writing data to a message object

```
AmBytes *dataSent = new AmBytes((const char*)"message to be sent");
mySendMessage->writeBytes(*dataSent);
```

Next, the message is sent using the sender service `mySender`.

Sending a message

```
mySender->send(*mySendMessage);
```

The policy used is either the default policy for the service, if specified, or the system default policy. The message attributes are set from the policy or service, or the default for the messaging transport.

When more control is needed, you can pass a policy object:

Sending a message with a specified policy

```
mySender->send(*mySendMessage, *myPolicy);
```

The policy controls the behavior of the send command. In particular, the policy specifies whether the send is part of a unit of work, the priority, persistence and expiry of the message and whether policy components should be invoked. Whether the queue should be implicitly opened and left open can also be controlled.

To send a message to a distribution list, for example `myDistList`, use it as the sender service:

Sending a message to a distribution list

```
myDistList->send(*mySendMessage);
```

You can set an attribute such as the *Format* before a message is sent, to override the default in the policy or service.

Setting an attribute in a message

```
mySendMessage->setFormat("MyFormat");
```

Similarly, after a message has been sent you can retrieve an attribute such as the *MessageID*. Binary data, such as *MessageId* can be extracted using the *AmBytes* helper class.

Getting an attribute from a message

```
AmBytes msgId = mySendMessage.getMessageId();
```

For details of the message attributes that you can set and get, see “*AmMessage*” on page 171.

When a message object is used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see “reset” on page 195) and rebuild it each time.

Sample program

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

Receiving messages

The next example shows how to receive a message from the receiver service `myReceiver`, and to read the data from the message object `myReceiveMessage`.

Receiving a message and retrieving the data

```
myReceiver->receive(*myReceiveMessage);  
AmBytes data = myReceiveMessage->readBytes(  
    myReceiveMessage->getDataLength());
```

The policy used will be the default for the service if defined, or the system default policy. Greater control of the behavior of the receive can be achieved by passing a policy object.

Receiving a message with a specified policy

```
myReceiver->receive(*myReceiveMessage, *myPolicy);
```

Writing applications in C++

The policy can specify the wait interval, whether the call is part of a unit of work, whether the message should be code page converted, whether all the members of a group must be there before any members can be read, and how to deal with backout failures.

To receive a specific message using its correlation ID, create a selection message object and set its *CorrelId* attribute to the required value. The selection message is then passed as a parameter on the receive.

Receiving a specific message using the correlation ID

```
AmBytes * myCorrelId = new AmBytes("MYCORRELATION");  
mySelectionMessage = mySession->createMessage("MY.SELECTION.MESSAGE");  
mySelectionMessage->setCorrelationId(*myCorrelId);  
myReceiver->receive(*myReceiveMessage, *mySelectionMessage, *myPolicy);
```

As before, the policy is optional.

You can view the attributes of the message just received, such as the *Encoding*.

Getting an attribute from the message

```
encoding = myReceiveMessage->getEncoding();
```

Sample program

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

Request/response messaging

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

In the following examples, ‘my’ refers to the requesting application (the client) and ‘your’ refers to the responding application (the server).

The requester sends a message as described in “Sending messages” on page 152, specifying the service (*myReceiver*) to which the response message should be sent.

Sending a request message

```
mySender->send(*mySendMessage, *myReceiver);
```

A policy object can also be specified if required.

The responder receives the message as described in “Receiving messages” on page 153, using its receiver service (*yourReceiver*). It also receives details of the response service (*yourResponder*) for sending the response.

Receiving the request message

```
yourReceiver->receive(*yourReceiveMessage, *yourResponder);
```

A policy object can be specified if required, as can a selection message object (see “Receiving messages” on page 153).

The responder sends its response message (`yourReplyMessage`) to the response service, specifying the received message to which this is a response.

Sending a response to the request message

```
yourResponder->send(*yourReplyMessage, *yourReceiveMessage);
```

Finally, the requester application receives the response (`myResponseMessage`), which is correlated with the original message it sent (`mySendMessage`).

Receiving the response message

```
myReceiver->receive(*myResponseMessage, *mySendMessage);
```

In a typical application, the responder might be a server operating in a loop, receiving requests and replying to them. In this case, the message objects should be set to their initial state and the data cleared before servicing the next request. This is achieved as follows:

Resetting the message object

```
yourReceiveMessage->reset();
yourResponseMessage->reset();
```

Sample programs

For more details, refer to the `Client.cpp` and `Server.cpp` sample programs (see “Sample programs for AS/400, UNIX, and Windows” on page 461).

File transfer

You can perform file transfers using the `AmSender.sendFile` and `AmReceiver.receiveFile` methods.

Sending a file using the sendFile method

```
mySender->sendFile(*mySendMessage, myfilename, *myPolicy)
```

Receiving a file using the receiveFile method

```
myReceiver->receiveFile(*myReceiveMessage, myfileName, *myPolicy)
```

For a complete description of file transfer, refer to “File transfer” on page 21.

Writing applications in C++

Publish/subscribe messaging

With *publish/subscribe* messaging, a *publisher* application publishes messages to *subscriber* applications using a *broker*. The message published contains application data and one or more *topic* strings that describe the data. A subscribing application subscribes to topics, informing the broker which topics it is interested in. When the broker receives a message from a publisher, it compares the topics in the messages to the topics in the subscription from subscribing applications. If they match, the broker forwards the message to the subscribing application.

Data on a particular topic is published as shown in the next example.

Publishing a message on a specified topic

```
AmBytes *publicationData = new AmBytes("The weather is sunny");

myPubMessage->addTopic("Weather");
myPubMessage->writeBytes(publicationData);
myPublisher->publish(*myPubMessage, *myReceiver);
```

`myReceiver` identifies a response service to which the broker will send any response messages (indicating whether the publish was successful or not). You can also specify a policy object to modify the behavior of the command.

To subscribe to a publish/subscribe broker you need to specify one or more topics.

Subscribing to a broker on specified topics

```
mySubMessage->addTopic("Weather");
mySubMessage->addTopic("Birds");
mySubscriber->subscribe(*mySubMessage, *myReceiver);
```

Broker response messages will be sent to `myReceiver`.

To remove a subscription, add the topic or topics to be deleted to the message object, and use:

Removing a subscription

```
mySubscriber->unsubscribe(*myUnsubMessage, *myReceiver);
```

To receive a publication from a broker, use:

Receiving a publication

```
mySubscriber->receive(*myReceiveMessage, *myPolicy);
publication = myReceiveMessage->readBytes(
    *myReceiveMessage->getDataLength());
```

You can then use the `getTopicCount` and `getTopic` methods to extract the topic or topics from the message object.

Subscribing applications can also exploit content-based publish/subscribe by passing a filter on subscribe and unsubscribe calls (see “Using MQSeries Integrator Version 2” on page 458).

Sample programs

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

Using AmElement objects

Publish/subscribe brokers (such as MQSeries Publish/Subscribe) respond to messages that contain name/value pairs to define the commands and options to be carried out. The Application Messaging Interface contains some methods which produce these name/value pairs directly (such as `AmSubscriber->subscribe`). For less commonly used commands, the name/value pairs can be added to a message using an `AmElement` object.

For example, to send a message containing a ‘Request Update’ command, use the following:

Using an AmElement object to construct a command message

```
AmElement *bespokeElement = new AmElement("MQPSCCommand", "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.

`AmC++` 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 `AmC++` 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`

Writing applications in C++

will be raised to reflect this fact. If this happens, the AmSender object will not be created, because its creation was interrupted by an Exception. However, there might be times during the life of an AmObject when processing AmWarningExceptions is useful.

For example:

```
try
{
    ...
    mySession->enableWarnings(AMB_TRUE);
    mySession->open();
    ...
}
catch (AmErrorException &errorEx)
{
    AmStatus sessionStatus = mySession->getLastErrorStatus();
    switch (sessionStatus.getReasonCode())
    {
        case AMRC_XXXX:
            ...
        case AMRC_XXXX:
            ...
    }
    mySession->clearErrorCodes();
}
catch (AmWarningException &warningEx)
{
    ...
}
```

Because most of the objects are types of AmObject, a generic error handling routine can be written. For example:

```
try
{
    ...
    mySession->open();
    ...
    mySender->send(*myMessage);
    ...
    mySender->send(*myMessage);
    ...
    mySession->commit();
}
catch(AmException &amex);
{
    AmStatus status = amex.getSource()->getLastErrorStatus();
    printf("Object in error; name = %s\n", amex.getSource()->getName());
    printf("Object in error; RC = %ld\n", status.getReasonCode());
    ...
    amex.getSource()->clearErrorCodes();
}
```

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

Transaction support

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

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

- MQSeries messages are the only resource
A transaction is started by the first message sent or received under syncpoint control, as specified in the policy specified for the send or receive. Multiple messages can be included in the same unit of work. The transaction is committed or backed out using the **commit** or **rollback** method.
- Using MQSeries as an XA transaction coordinator
The transaction must be started explicitly using the **begin** method before the first recoverable resource (such as a relational database) is changed. The transaction is committed or backed out using an **commit** or **rollback** method.
- Using an external transaction coordinator
The transaction is controlled using the API calls of an external transaction coordinator (such as CICS, Encina or Tuxedo). The AMI calls are not used but the syncpoint attributed must still be specified in the policy used on the call.

Sending group messages

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

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

The message status is set using the **AmMessage.setGroupStatus** method.

For a complete description of group messages, refer to “Sending group messages” on page 26

Other considerations

You should also consider the following.

Multithreading

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

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

Using MQSeries with the AMI

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

Writing applications in C++

Field limits

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

Building C++ applications

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

AMI include files

AMI provides include files, **amtc.h** and **amtcpp.hpp**, to assist you with the writing of your applications. It is recommended that you become familiar with the contents of these files.

The include files are installed under:

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

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

Your AMI C++ program must contain the statement:

```
#include <amtcpp.hpp>
```

Even though you need mention only the C++ include file, both **amtc.h** and **amtcpp.hpp** must be accessible to your program at compilation time.

Next step

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

- “C++ applications on AIX”
- “C++ applications on AS/400” on page 162
- “C++ applications on HP-UX” on page 163
- “C++ applications on Solaris” on page 164
- “C++ applications on Windows” on page 165

C++ applications on AIX

This section explains what you have to do to prepare and run your C++ programs on the AIX operating system. See “Language compilers” on page 426 for the compilers supported by the AMI.

Preparing C++ programs on AIX

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

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

- Where the AMI include files are.

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

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 `-l` flag, more specifically `-lamtCpp`.

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

```
xlc -I/usr/mqm/amt/inc -L/usr/mqm/lib -lamtCpp mine.cpp -o mine
```

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

```
xlc_r -I/usr/mqm/amt/inc -L/usr/mqm/lib -lamtCpp_r mine.cpp -o mine
```

Running C++ programs on AIX

To run a C++ executable, you must have access to the C++ library `libamtCpp.a` in your runtime environment. If the `amtInstall` utility has been run, this environment will be set up for you (see “Installation on AIX” on page 428).

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

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

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

```
ln -s /usr/mqm/lib/libamtCpp_r.a /usr/lib/libamtCpp_r.a
```

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

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

For the non-threaded MQSeries Client library, perform:

```
ln -s /usr/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 426 for the compilers supported by the AMI.

Preparing C++ programs on AS/400

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

To compile a C++ module using the ILE compiler, you can use the OS/400 command `CRTCPPMOD`. The library `QMQMAMI` must be in the library list because it contains the `amtcpp.hpp` header file.

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 **BNSRVPGM** option of **CRTPGM**. For example:

```
CRTPGM PGM(pgmname) MODULE(pgmname) BNSRVPGM(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 BNSRVPGM 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 426 for the compilers supported by the AMI.

Preparing C++ programs on HP-UX

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

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

1. Where the AMI include files are.

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

2. Where the AMI libraries are.

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

3. Link with the AMI library for C++.

To do this, use the **-l** flag, more specifically **-lamtCpp**.

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

```
aCC +DAportable -Wl,+b,.-L/opt/mqm/lib -o mine mine.cpp
-I/opt/mqm/amt/inc -lamtCpp
```

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

Running C++ programs on HP-UX

To run a C++ executable, you must have access to the C++ library `libamtCpp.sl` in your runtime environment. If **amtInstall** utility has been run, this environment will be set up for you (see “Installation on HP-UX” on page 436).

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

```
ln -s /opt/mqm/lib/libamtCpp_r.sl /usr/lib/libamtCpp.sl
```

C++ applications on HP-UX

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

```
ln -s /opt/mqm/lib/libamtCpp_r.sl /usr/lib/libamtCpp_r.sl
```

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

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

For the non-threaded MQSeries Client library, perform:

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

For the threaded MQSeries Server library, perform:

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

For the threaded MQSeries Client library, perform:

```
ln -s /opt/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 Solaris

This section explains what you have to do to prepare and run your C++ programs in the Sun Solaris operating environment. See “Language compilers” on page 426 for the compilers supported by the AMI.

Preparing C++ programs on Solaris

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

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

- Where the AMI include files are.
To do this, use the `-I` flag. In the case of Solaris, they are usually located at `/opt/mqm/amt/inc`.
- Where the AMI library is.
To do this, use the `-L` flag. In the case of Solaris, it is usually located at `/opt/mqm/lib`.
- Link with the AMI library.
To do this, use the `-l` flag, more specifically `-lamtCpp`.

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

```
CC -mt -I/opt/mqm/amt/inc -L/opt/mqm/lib -lamtCpp mine.cpp -o mine
```

Running C++ programs on Solaris

To run a C++ executable, you must have access to the C++ library `libamtCpp.so` in your runtime environment. If the `amtInstall` utility has been run, this environment will be set up for you (see “Installation on Sun Solaris” on page 443).

If you have not run the utility, the easiest way to set up the required access is to construct a link from the Solaris default library location to the actual location of the C++ libraries.

To do this, enter:

```
ln -s /opt/mqm/lib/libamtCpp.so /usr/lib/libamtCpp.so
```

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

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

For the MQSeries Client library, perform:

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

C++ applications on Windows

This section explains what you have to do to prepare and run your C++ programs on the Windows 98 and Windows NT operating systems. See “Language compilers” on page 426 for the compilers supported by the AMI.

Preparing C++ programs on Windows

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

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

1. Where the AMI include files are.

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

2. Where the AMI library is.

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

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

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

Running C++ programs on Windows

To run a C++ executable, you must have access to the C++ DLL `amtCpp.dll` in your runtime environment. Make sure it exists in one of the directories pointed to by the `PATH` environment variable. For example:

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

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

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

C++ applications on Windows

Chapter 7. C++ interface overview

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

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

Base classes

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

Helper classes

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

Exception classes

AmException	page 181
AmErrorException	page 181
AmWarningExcpetion	page 181

AmSessionFactory

The `AmSessionFactory` class is used to create `AmSession` objects.

Constructor

Constructor for `AmSessionFactory`.

`AmSessionFactory` page 184

Session factory management

Methods to return the name of an `AmSessionFactory` object, to get and set the names of the AMI data files (local host and repository), and to control traces.

`getFactoryName` page 184

`getLocalHost` page 184

`getRepository` page 184

`getTraceLevel` page 184

`getTraceLocation` page 184

`setLocalHost` page 185

`setRepository` page 185

`setTraceLevel` page 185

`setTraceLocation` page 185

Create and delete session

Methods to create and delete an `AmSession` object.

`createSession` page 184

`deleteSession` page 184

AmSession

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

Session management

Methods to open and close an AmSession object, to return its name, and to control traces.

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

Create objects

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

createMessage	page 187
createSender	page 187
createReceiver	page 187
createDistributionList	page 186
createPublisher	page 187
createSubscriber	page 187
createPolicy	page 187

Delete objects

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

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

Transactional processing

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

begin	page 186
commit	page 186
rollback	page 189

C++ interface overview

Error handling

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

clearErrorCodes page 186

enableWarnings page 188

getLastErrorStatus page 189

AmMessage

An **AmMessage** object encapsulates an MQSeries message descriptor (MQMD) structure, and contains the message data.

Get values

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

getCCSID	page 192
getCorrelationId	page 192
getElementCCSID	page 193
getEncoding	page 193
getFormat	page 193
getGroupStatus	page 194
getMessageId	page 194
getName	page 194
getReportCode	page 195
getType	page 195

Set values

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

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

Reset values

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

reset	page 195
--------------	----------

Read and write data

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

getDataLength	page 192
getDataOffset	page 192
setDataOffset	page 196
readBytes	page 195
writeBytes	page 197

C++ interface overview

Publish/subscribe topics

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

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

Publish/subscribe filters

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

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

Publish/subscribe name/value elements

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

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

Error handling

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

clearErrorCodes	page 191
enableWarnings	page 192
getLastErrorStatus	page 194

AmSender

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

Open and close

Methods to open and close the sender service.

open	page 199
close	page 198

Send

Method to send a message.

send	page 199
-------------	----------

Send file

Method to send data from a file

sendFile	page 200
-----------------	----------

Get values

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

getCCSID	page 198
getEncoding	page 198
getName	page 199

Error handling

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

clearErrorCodes	page 198
enableWarnings	page 198
getLastErrorStatus	page 199

AmReceiver

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

Open and close

Methods to open and close the receiver service.

open	page 203
close	page 202

Receive and browse

Methods to receive or browse a message.

receive	page 203
browse	page 201

Receive file

Method to receive file message data into a file.

receiveFile	page 204
--------------------	----------

Get values

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

getDefinitionType	page 202
getName	page 203
getQueueName	page 203

Set value

Method to set the queue name of the receiver service.

setQueueName	page 204
---------------------	----------

Error handling

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

clearErrorCodes	page 202
enableWarnings	page 202
getLastErrorStatus	page 203

AmDistributionList

An **AmDistributionList** object encapsulates a list of **AmSender** objects.

Open and close

Methods to open and close the distribution list service.

open page 205

close page 205

Send

Method to send a message to the distribution list.

send page 206

Send file

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

sendFile page 206

Get values

Methods to get the name of the distribution list service, a count of the **AmSenders** in the list, and one of the **AmSenders** that is contained in the list.

getName page 205

getSenderCount page 205

getSender page 205

Error handling

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

clearErrorCodes page 205

enableWarnings page 205

getLastErrorStatus page 205

AmPublisher

An **AmPublisher** object encapsulates a sender service and provides support for publishing messages to a publish/subscribe broker.

Open and close

Methods to open and close the publisher service.

open	page 208
close	page 207

Publish

Method to publish a message.

publish	page 208
----------------	----------

Get values

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

getCCSID	page 207
getEncoding	page 207
getName	page 207

Error handling

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

clearErrorCodes	page 207
enableWarnings	page 207
getLastErrorStatus	page 207

AmSubscriber

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

Open and close

Methods to open and close the subscriber service.

open	page 210
close	page 209

Broker messages

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

subscribe	page 211
unsubscribe	page 211
receive	page 210

Get values

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

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

Set value

Method to set the queue name of the subscriber service.

setQueueName	page 210
---------------------	----------

Error handling

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

clearErrorCodes	page 209
enableWarnings	page 209
getLastErrorStatus	page 209

AmPolicy

An **AmPolicy** object encapsulates the options used during AMI operations.

Policy management

Methods to return the name of the policy, and to get and set the wait time when receiving a message.

getName	page 212
getWaitTime	page 212
setWaitTime	page 212

Error handling

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

clearErrorCodes	page 212
enableWarnings	page 212
getLastErrorStatus	page 212

Helper classes

The classes that encapsulate name/value elements for publish/subscribe, strings, binary data and error status.

AmBytes

The AmBytes class is an encapsulation of a byte array. It allows the AMI to pass byte strings across the interface and enables manipulation of byte strings. It contains constructors, operators and a destructor, and methods to copy, compare, and pad. AmBytes also has methods to give the length of the encapsulated bytes and a method to reference the data contained within an AmBytes object.

constructors	page 213
destructor	page 214
operators	page 214
cmp	page 213
cpy	page 214
dataPtr	page 214
length	page 214
pad	page 214

AmElement

Constructor for AmElement, and methods to return the name, type, value and version of an element, to set the version, and to return an AmString representation of the element.

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

AmObject

A virtual class containing methods to return the name of the object, to clear the error codes and to return the last error condition.

clearErrorCodes	page 216
getLastErrorStatus	page 216
getName	page 216

AmStatus

Constructor for AmStatus, and methods to return the completion code, reason code, secondary reason code and status text, and to return an AmString representation of the AmStatus.

AmStatus	page 217
getCompletionCode	page 217

C++ interface overview

getReasonCode	page 217
getReasonCode2	page 217
toString	page 217

AmString

The AmString class is an encapsulation of a string. It allows the AMI to pass strings across the interface and enables manipulation of strings. It contains constructors, operators, a destructor, and methods to copy, concatenate, pad, split, truncate and strip. AmString also has methods to give the length of the encapsulated string, compare AmStrings, check whether one AmString is contained within another and a method to reference the text of an AmString.

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

Exception classes

Classes that encapsulate error and warning conditions. `AmErrorException` and `AmWarningException` inherit from `AmException`.

AmException

Methods to return the completion code and reason code from the `Exception`, the class name, method name and source of the `Exception`, and to return a string representation of the `Exception`.

<code>getClassname</code>	page 220
<code>getCompletionCode</code>	page 220
<code>getMethodname</code>	page 220
<code>getReasonCode</code>	page 220
<code>getSource</code>	page 220
<code>toString</code>	page 220

AmErrorException

Methods to return the completion code and reason code from the `Exception`, the class name, method name and source of the `Exception`, and to return a string representation of the `Exception`.

<code>getClassname</code>	page 221
<code>getCompletionCode</code>	page 221
<code>getMethodname</code>	page 221
<code>getReasonCode</code>	page 221
<code>getSource</code>	page 221
<code>toString</code>	page 221

AmWarningException

Methods to return the completion code and reason code from the `Exception`, the class name, method name and source of the `Exception`, and to return a string representation of the `Exception`.

<code>getClassname</code>	page 222
<code>getCompletionCode</code>	page 222
<code>getMethodname</code>	page 222
<code>getReasonCode</code>	page 222
<code>getSource</code>	page 222
<code>toString</code>	page 222

C++ interface overview

Chapter 8. C++ interface reference

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

Base classes

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

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

Helper classes

<code>AmBytes</code>	page 213
<code>AmElement</code>	page 215
<code>AmObject</code>	page 216
<code>AmStatus</code>	page 217
<code>AmString</code>	page 218

Exception classes

<code>AmException</code>	page 220
<code>AmErrorException</code>	page 221
<code>AmWarningException</code>	page 222

AmSessionFactory

The **AmSessionFactory** class is used to create **AmSession** objects.

AmSessionFactory

Constructors for an **AmSessionFactory**.

```
AmSessionFactory();  
AmSessionFactory(char * name);
```

name The name of the **AmSessionFactory**. This is the location of the data files used by the AMI (the repository file and the local host file). The name should be a fully qualified directory that includes the path under which the files are located. Otherwise, see "Local host and repository files (AS/400, UNIX, and Windows)" on page 450 for the location of these files.

createSession

Creates an **AmSession** object.

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

name The name of the **AmSession**.

deleteSession

Deletes an **AmSession** object previously created using the **createSession** method.

```
void deleteSession(AmSession ** pSession);
```

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

getFactoryName

Returns the name of the **AmSessionFactory**.

```
AmString getFactoryName();
```

getLocalHost

Returns the name of the local host file.

```
AmString getLocalHost();
```

getRepository

Returns the name of the repository file.

```
AmString getRepository();
```

getTraceLevel

Returns the trace level for the **AmSessionFactory**.

```
int getTraceLevel();
```

getTraceLocation

Returns the location of the trace for the **AmSessionFactory**.

```
AmString getTraceLocation();
```

setLocalHost

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

```
void setLocalHost(char * fileName);
```

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

setRepository

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

```
void setRepository(char * fileName);
```

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

setTraceLevel

Sets the trace level for the AmSessionFactory.

```
void setTraceLevel(int level);
```

level The trace level to be set in the AmSessionFactory. Trace levels are 0 through 9, where 0 represents minimal tracing and 9 represents a fully detailed trace.

setTraceLocation

Sets the location of the trace for the AmSessionFactory.

```
void setTraceLocation(char * location);
```

location The location on the local system where trace files will be written. This location must be a directory, and it must exist before the trace is run.

AmSession

An **AmSession** object provides the scope for a unit of work and creates and manages all other objects, including at least one connection object. Each (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').

C++ AmSession

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);
```

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

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmSession.

```
String getName();
```

getTraceLevel

Returns the trace level of the AmSession.

```
int getTraceLevel();
```

getTraceLocation

Returns the location of the trace for the AmSession.

```
AmString getTraceLocation();
```

open

Opens an AmSession using the specified policy. The application profile group of this policy provides the connection definitions enabling the connection objects to be created. The specified library is loaded for each connection and its dispatch table initialized. If the transport type is MQSeries and the MQSeries local queue manager library cannot be loaded, the MQSeries client queue manager is loaded. Each connection object is then opened.

open is overloaded: the policy parameter is optional.

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

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

rollback

Rolls back a unit of work that was started by **AmSession.begin**, or under policy control. **rollback** is overloaded: the policy parameter is optional.

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

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

AmMessage

An **AmMessage** object encapsulates the MQSeries MQMD message properties, and name/value elements such as the topics for publish/subscribe messages. In addition it contains the application data.

The initial state of the message object is:

CCSID	default queue manager CCSID
correlationId	all zeros
dataLength	zero
dataOffset	zero
elementCount	zero
encoding	AMENC_NATIVE
format	AMFMT_STRING
groupStatus	AMGRP_MSG_NOT_IN_GROUP
reportCode	AMFBP_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 195) and rebuild it each time.

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

addElement	page 191
deleteElement	page 191
getElement	page 193
getElementCount	page 193
deleteNamedElement	page 192
getNamedElement	page 194
getNamedElementCount	page 194
addTopic	page 191
deleteTopic	page 192
getTopic	page 195
getTopicCount	page 195

addElement

Adds a name/value element to an AmMessage object. **addElement** is overloaded: the `element` parameter is required, but the `options` parameter is optional.

```
void addElement(
    AmElement &element,
    int        options);
```

element The element to be added to the AmMessage.

options The options to be used. This parameter is reserved and must be set to zero.

addFilter

Adds a publish/subscribe filter to an AmMessage object.

```
void addFilter(char * filter);
```

filter The filter to be added to the AmMessage.

addTopic

Adds a publish/subscribe topic to an AmMessage object.

```
void addTopic(char * topicName);
```

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

clearErrorCodes

Clears the error in the AmMessage object.

```
void clearErrorCodes();
```

deleteElement

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

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

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

getElementCount gets the number of elements in the message.

deleteFilter

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

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

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

C++ AmMessage

deleteNamedElement

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

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

name The name of the element to be deleted.

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

getNamedElementCount gets the number of elements in the message with the specified name.

deleteTopic

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

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

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

enableWarnings

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

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

warningsOn If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier used by the AmMessage.

```
int getCCSID();
```

getCorrelationId

Returns the correlation identifier for the AmMessage.

```
AmBytes getCorrelationId();
```

getDataLength

Returns the length of the message data in the AmMessage.

```
int getDataLength();
```

getDataOffset

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

```
int getDataOffset();
```

getElement

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

```
AmElement getElement(int index);
```

index The index of the element to be returned, starting from zero. **getElementCount** gets the number of elements in the message.

getElementCCSID

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

```
int getElementCCSID();
```

getElementCount

Returns the total number of elements in an AmMessage object. This might include topics (which are specialized elements).

```
int getElementCount();
```

getEncoding

Returns the value used to encode numeric data types for the AmMessage.

```
int getEncoding();
```

The following values can be returned:

```
AMENC_NATIVE
AMENC_NORMAL
AMENC_NORMAL_FLOAT_390
AMENC_REVERSED
AMENC_REVERSED_FLOAT_390
AMENC_UNDEFINED
```

getFilter

Returns the publish/subscribe filter in the AmMessage object at the specified index. Indexing is within all filters.

```
AmString getFilter(int filterIndex);
```

filterIndex The index of the filter to be returned, starting from zero. **getElementCount** gets the number of filters in a message.

getFilterCount

Returns the total number of publish/subscribe filters in the AmMessage object.

```
AmElement getFilterCount();
```

getFormat

Returns the format of the AmMessage.

```
AmString getFormat();
```

The following values can be returned:

```
AMFMT_NONE
AMFMT_STRING
AMFMT_RF_HEADER
```

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();
```

The following values can be returned:

```
AMFB_NONE
AMFB_EXPIRATION
AMFB_COA
AMFB_COD
AMFB_ERROR
```

getTopic

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

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

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

getTopicCount

Returns the total number of publish/subscribe topics in the AmMessage object.

```
int getTopicCount();
```

getType

Returns the message type from the AmMessage.

```
int getType();
```

The following values can be returned:

```
AMMT_REQUEST
AMMT_REPLY
AMMT_REPORT
AMMT_DATAGRAM
```

readBytes

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

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

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

reset

Resets the AmMessage object to its initial state (see page 190).

reset is overloaded: the options parameter is optional.

```
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 a null string; for example:

```
myMessage.setCorrelationId(AmBytes(""));
```

setDataOffset

Sets the data offset for reading or writing byte data.

```
void setDataOffset(int dataOffset);
```

dataOffset

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

setElementCCSID

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

```
void setElementCCSID(int elementCCSID);
```

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).

setGroupStatus

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

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

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

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

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

```
AMGRP_MSG_NOT_IN_GROUP
AMGRP_FIRST_MSG_IN_GROUP
AMGRP_MIDDLE_MSG_IN_GROUP
AMGRP_LAST_MSG_IN_GROUP
AMGRP_ONLY_MSG_IN_GROUP
```

writeBytes

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

```
void writeBytes(AmBytes &data);
```

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

AmSender

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

The AmSender object must be created before it can be opened. This is done using **AmSession.createSender**.

A *responder* is a special type of AmSender used for sending a response to a request message. It is not created from a repository definition. Once created, it must not be opened until used in its correct context as a responder receiving a request message with **AmReceiver.receive**. When opened, its queue and queue manager properties are modified to reflect the *ReplyTo* destination specified in the message being received. When first used in this context, the sender service becomes a responder sender service.

clearErrorCodes

Clears the error codes in the AmSender.

```
void clearErrorCodes();
```

close

Closes the AmSender. **close** is overloaded: the *policy* parameter is optional.

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

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

enableWarnings

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

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

warningsOn If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getCCSID

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

```
int getCCSID();
```

getEncoding

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

```
int getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmSender.

```
AmString getName();
```

open

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

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

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

send

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

send is overloaded: the sendMessage parameter is required, but the others are optional. receivedMessage and responseService are used in request/response messaging, and are mutually exclusive.

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

sendMessage The message object that contains the data to be sent.

responseService

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

receivedMessage

The previously received message which is used for correlation with the sent message. If omitted, the sent message is not correlated with any received message.

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

C++ AmSender

sendFile

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

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

- sendMessage** The message object to use to send the file. This can be used to specify the Correlation ID for example.
- fileName** The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with the receive file method (see “`receiveFile`” on page 204 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
- policy** The policy to be used. If omitted, the system default policy (name constant : `AMSD_POL`) is used.

AmReceiver

An **AmReceiver** object encapsulates an MQSeries object descriptor (MQOD) structure. This represents an MQSeries queue on a local or remote queue manager. An open AmReceiver is always associated with an open connection object, such as a queue manager connection. Support is also included for a dynamic AmReceiver (that encapsulates a model queue). The required AmReceiver object definitions can be provided from a repository or can be created automatically from the set of existing queue objects available on the local queue manager.

There is a definition type associated with each AmReceiver:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

An AmReceiver created from a repository definition will be initially of type AMDT_PREDEFINED or AMDT_DYNAMIC. When opened, its definition type might change from AMDT_DYNAMIC to AMDT_TEMP_DYNAMIC according to the properties of its underlying queue object.

An AmReceiver created with default values (that is, without a repository definition) will have its definition type set to AMDT_UNDEFINED until it is opened. When opened, this will become AMDT_DYNAMIC, AMDT_TEMP_DYNAMIC, or AMDT_PREDEFINED, according to the properties of its underlying queue object.

browse

Browses an AmReceiver service. **browse** is overloaded: the browseMessage and options parameters are required, but the others are optional.

```
void browse(
    AmMessage &browseMessage,
    int options,
    AmSender &responseService,
    AmMessage &selectionMessage,
    AmPolicy &policy);
```

browseMessage The message object that receives the browse data.

options Options controlling the browse operation. Possible values are:

```
AMBRW_NEXT
AMBRW_FIRST
AMBRW_CURRENT
AMBRW_RECEIVE_CURRENT
AMBRW_DEFAULT (AMBRW_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.

C++ AmReceiver

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 196) before invoking the browse method.

policy

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

clearErrorCodes

Clears the error codes in the AmReceiver.

```
void clearErrorCodes();
```

close

Closes the AmReceiver. **close** is overloaded: the policy parameter is optional.

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

policy

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

enableWarnings

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

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

warningsOn

If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getDefinitionType

Returns the definition type (service type) for the AmReceiver.

```
int getDefinitionType();
```

The following values can be returned:

```
AMDT_UNDEFINED  
AMDT_TEMP_DYNAMIC  
AMDT_DYNAMIC  
AMDT_PREDEFINED
```

Values other than AMDT_UNDEFINED reflect the properties of the underlying queue object.

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmReceiver.

```
AmString getName();
```

getQueueName

Returns the queue name of the AmReceiver. This is used to determine the queue name of a permanent dynamic AmReceiver, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **setQueueName**.)

```
AmString getQueueName();
```

open

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

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

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

receive

Receives a message from the AmReceiver service. **receive** is overloaded: the receiveMessage parameter is required, but the others are optional.

```
void receive(
    AmMessage &receiveMessage,
    AmSender &responseService,
    AmMessage &selectionMessage,
    AmPolicy &policy);
```

receiveMessage

The message object that receives the data. The message object is reset implicitly before the receive takes place.

responseService

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

selectionMessage

A message object containing the Correlation ID used to selectively receive a message from the AmReceiver. If omitted, the first available message is received. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be reset (see **AmMessage.setCorrelationId** on page 196) before invoking the receive method.

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

C++ AmReceiver

receiveFile

Receives file message data into a file. To receive data into a file, the `receiveMessage` parameter is required, but the others are optional.

```
void receiveFile(  
    AmMessage &receiveMessage,  
    char *    &fileName,  
    AmMessage &selectionMessage,  
    AmPolicy  &policy);
```

receiveMessage

The message object used to receive the file. This is updated with the message properties, for example the Message ID. If the message is not from a file, the message object receives the data. The message object is reset implicitly before the receive takes place.

fileName

The name of the file to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If NULL or a null string is specified, the AMI will use the name of the originating file (including any directory prefix), exactly as it was supplied on the send file call. Note that the original file name may not be appropriate for use by the receiver, either because a path name included in the file name is not applicable to the receiving system, or because the sending and receiving systems use different file naming conventions.

selectionMessage

A message object containing the Correlation ID used to selectively receive a message from the AmReceiver. If omitted, the first available message is received. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be reset (see **AmMessage.setCorrelationId** on page 196) before invoking the receive method.

policy

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

setQueueName

Sets the queue name of the AmReceiver (when this encapsulates a model queue). This is used to specify the queue name of a recreated permanent dynamic AmReceiver, in order to receive messages in a session subsequent to the one in which it was created. (See also **getQueueName**.)

```
void setQueueName(char * queueName);
```

queueName

The queue name to be set in the AmReceiver.

AmDistributionList

An **AmDistributionList** object encapsulates a list of AmSender objects.

clearErrorCodes

Clears the error codes in the AmDistributionList.

```
void clearErrorCodes();
```

close

Closes the AmDistributionList. **close** is overloaded: the `policy` parameter is optional.

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

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

enableWarnings

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

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

warningsOn If set to `AMB_TRUE`, AmWarningExceptions will be raised for this object.

getLastErrorStatus

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

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmDistributionList object.

```
AmString getName();
```

getSender

Returns a pointer to the AmSender object contained within the AmDistributionList object at the index specified. `AmDistributionList.getSenderCount` gets the number of AmSender services in the distribution list.

```
AmSender * getSender(int index);
```

index The index of the AmSender in the AmDistributionList, starting at zero.

getSenderCount

Returns the number of AmSender services in the AmDistributionList object.

```
int getSenderCount();
```

open

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

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

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

C++ AmDistributionList

send

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

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

sendMessage The message object containing the data to be sent.

responseService

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

policy

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

sendFile

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

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

sendMessage The message object to use to send the file. This can be used to specify the Correlation ID, for example. The message must not include any elements or data.

fileName

The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with the receive file method (see “receiveFile” on page 204 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.

policy

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

AmPublisher

An **AmPublisher** object encapsulates an AmSender and provides support for publish requests to a publish/subscribe broker.

clearErrorCodes

Clears the error codes in the AmPublisher.

```
void clearErrorCodes();
```

close

Closes the AmPublisher. **close** is overloaded: the `policy` parameter is optional.

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

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

enableWarnings

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

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

warningsOn If set to `AMB_TRUE`, AmWarningExceptions will be raised for this object.

getCCSID

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

```
int getCCSID();
```

getEncoding

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

```
int getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmPublisher.

```
AmString getName();
```


C++ AmPublisher

open

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

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

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

publish

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

```
void publish(  
    AmMessage &pubMessage,  
    AmReceiver &responseService,  
    AmPolicy &policy);
```

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

responseService

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

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

AmSubscriber

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

clearErrorCodes

Clears the error codes in the AmSubscriber.

```
void clearErrorCodes();
```

close

Closes the AmSubscriber. **close** is overloaded: the policy parameter is optional.

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

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

enableWarnings

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

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

warningsOn If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier for the AmSender in the AmSubscriber. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the subscriber must perform CCSID conversion of the message before it is sent.

```
int getCCSID();
```

getDefinitionType

Returns the definition type for the AmReceiver in the AmSubscriber.

```
int getDefinitionType();
```

The following values can be returned:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

getEncoding

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

```
int getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

C++ AmSubscriber

getName

Returns the name of the AmSubscriber.

```
AmString getName();
```

getQueueName

Returns the queue name used by the AmSubscriber to receive messages. This is used to determine the queue name of a permanent dynamic AmReceiver in the AmSubscriber, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **setQueueName**.)

```
AmString getQueueName();
```

open

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

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

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

receive

Receives a message, normally a publication, using the AmSubscriber. The message data, topic and other elements can be accessed using the message interface methods (see page 190).

receive is overloaded: the pubMessage parameter is required, but the others are optional.

```
void receive(  
    AmMessage &pubMessage,  
    AmMessage &selectionMessage,  
    AmPolicy &policy);
```

pubMessage The message object containing the data that has been published. The message object is reset implicitly before the receive takes place.

selectionMessage

A message object containing the correlation ID used to selectively receive a message from the AmSubscriber. If omitted, the first available message is received. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be reset (see **AmMessage.setCorrelationId** on page 196) before invoking the receive method.

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

setQueueName

Sets the queue name in the AmReceiver of the AmSubscriber, when this encapsulates a model queue. This is used to specify the queue name of a recreated permanent dynamic AmReceiver, in order to receive messages in a session subsequent to the one in which it was created. (See also **getQueueName**.)

```
void setQueueName(char * queueName);
```

queueName The queue name to be set.

subscribe

Sends a subscribe message to a publish/subscribe broker using the AmSubscriber, to register a subscription. The topic and other elements can be specified using the message interface methods (see page 190) before sending the message.

Publications matching the subscription are sent to the AmReceiver associated with the AmSubscriber. By default, this has the same name as the AmSubscriber, with the addition of the suffix '.RECEIVER'.

subscribe is overloaded: the subMessage parameter is required, but the others are optional.

```
void subscribe(
    AmMessage &subMessage,
    AmReceiver &responseService,
    AmPolicy &policy);
```

subMessage The message object that contains the topic subscription data.

responseService

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

This is not the AmReceiver to which publications will be sent by the broker; they are sent to the AmReceiver associated with the AmSubscriber (see above).

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

unsubscribe

Sends an unsubscribe message to a publish/subscribe broker using the AmSubscriber, to deregister a subscription. The topic and other elements can be specified using the message interface methods (see page 190) before sending the message.

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

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

unsubMessage The message object that contains the topics to which the unsubscribe request applies.

responseService

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

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

AmPolicy

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

clearErrorCodes

Clears the error codes in the AmPolicy.

```
void clearErrorCodes();
```

enableWarnings

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

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

warningsOn If set to AMB_TRUE, AmWarningExceptions will be raised for this object.

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmPolicy object.

```
AmString getName();
```

getWaitTime

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

```
int getWaitTime();
```

setWaitTime

Sets the wait time for any **receive** using this AmPolicy.

```
void setWaitTime(int waitTime);
```

waitTime The wait time (in ms) to be set in the AmPolicy.

AmBytes

An **AmBytes** object encapsulates an array of bytes. It allows the AMI to pass bytes across the interface and enables manipulation of these bytes.

cmp

Methods used to compare AmBytes objects. These methods return 0 if the data is the same, and 1 otherwise.

```
AMLONG cmp(const AmBytes &amBytes);
AMLONG cmp(const char * stringData);
AMLONG cmp(const char * charData, AMLONG length);
```

amBytes	A reference to the AmBytes object being compared.
stringData	A char pointer to the NULL terminated string being compared.
charData	A char pointer to the bytes being compared.
length	The length, in bytes, of the data to be compared. If this length is not the same as the length of the AmBytes object, the comparison fails.

constructors

Constructors for an AmBytes object.

```
AmBytes();
AmBytes(const AmBytes &amBytes);
AmBytes(const AMBYTE byte);
AmBytes(const AMLONG long);
AmBytes(const char * charData);
AmBytes(const AmString &amString);
AmBytes(const AMSTR stringData);
AmBytes(const AMBYTE *character, const AMLONG length);
```

amBytes	A reference to an AmBytes object used to create the new AmBytes object.
byte	A single byte used to create the new AmBytes object.
long	An AMLONG used to create the new AmBytes object.
charData	A char pointer to a NULL terminated string used to create the new AmBytes object.
stringData	A NULL terminated string used to create the new AmBytes object.
character	The character to populate the new AmBytes object with.
length	The length, in bytes, of the new AmBytes object.

C++ AmBytes

cpy

Methods used to copy from an AmBytes object. Any existing data in the AmBytes object is discarded.

```
AmBytes &cpy();  
AmBytes &cpy(const AMSTR stringData);  
AmBytes &cpy(const AMBYTE *byteData, const AMLONG length);  
AmBytes &cpy(const AMBYTE byte);  
AmBytes &cpy(const AMLONG long);  
AmBytes &cpy(const AmBytes &amBytes);
```

stringData A NULL terminated string being copied.
byteData A pointer to the bytes being copied.
length The length, in bytes, of the data to be copied.
byte The single byte being copied.
long An AMLONG being copied.
amBytes A reference to the AmBytes object being copied.

dataPtr

Method to reference the byte data contained within an AmBytes object.

```
const AMBYTE * dataPtr() const;
```

destructor

Destructor for an AmBytes object.

```
~AmBytes();
```

length

Returns the length of an AmBytes object.

```
AMLONG length();
```

operators

Operators for an AmBytes object.

```
AmBytes &operator = (const AmBytes &);  
AMBOOL operator == (const AmBytes &) const;  
AMBOOL operator != (const AmBytes &) const;
```

pad

Method used to pad AmBytes objects with a specified byte value.

```
AmBytes &pad(const AMLONG length, const AMBYTE byte);
```

length The required length of the AmBytes after the padding.

byte The byte value used to pad the AmBytes object.

AmElement

An **AmElement** object encapsulates a name/value pair which can be added to an AmMessage object.

AmElement

Constructor for an AmElement object.

```
AmElement(char * name, char * value);
```

name The name of the element.

value The value of the element.

getName

Returns the name of the AmElement.

```
AmString getName();
```

getValue

Returns the value of the AmElement.

```
AmString getValue();
```

getVersion

Returns the version of the AmElement (the default value is AMELEM_VERSION_1).

```
int getVersion();
```

setVersion

Sets the version of the AmElement.

```
void setVersion(int version);
```

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

toString

Returns a AmString representation of the AmElement.

```
AmString toString();
```


AmObject

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

- AmSession
- AmMessage
- AmSender
- AmDistributionList
- AmReceiver
- AmPublisher
- AmSubscriber
- AmPolicy

This allows application programmers to use generic error handling routines.

clearErrorCodes

Clears the error codes in the AmObject.

```
void clearErrorCodes();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmObject.

```
AmString getName();
```

AmStatus

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

AmStatus

Constructor for an AmStatus object.

```
AmStatus();
```

getCompletionCode

Returns the completion code from the AmStatus object.

```
int getCompletionCode();
```

getReasonCode

Returns the reason code from the AmStatus object.

```
int getReasonCode();
```

getReasonCode2

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

cpy

Methods used to copy from an AmString. Any existing data in the AmString is discarded.

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

amString A reference to an AmString object being copied.

stringData The NULL terminated string being copied into the AmString.

destructor

Destructor for an AmString object.

```
~AmString();
```

operators

Operators for an AmString object.

```
AmString &operator = (const AmString &);
AmString &operator = (const AMSTR);
AMBOOL operator == (const AmString &) const;
AMBOOL operator != (const AmString &) const;
```

pad

Method used to pad AmStrings with a specified character.

```
AmString &pad(const AMLONG length, const AMBYTE character);
```

length The required length of the AmString after the padding.

charString The character used to pad the AmString.

split

Method used to split AmStrings at the first occurrence of a specified character.

```
AmString &split(AmString &newString, const AMBYTE splitCharacter);
```

newString A reference to an AmString object to contain the latter half of the split string.

splitCharacter The first character at which the split will occur.

strip

Method used to strip leading and trailing blanks from AmStrings.

```
AmString &strip();
```

length

Returns the length of an AmString.

```
AMLONG length();
```

text

Method to reference the string contained within an AmString.

```
AMSTR text() const;
```

truncate

Method used to truncate AmStrings.

```
AmString &truncate(const AMLONG length);
```

length The length to which the AmString is to be truncated.

AmException

AmException is the base Exception class; all other Exceptions inherit from this class.

getClassName

Returns the type of object throwing the Exception.

```
AmString getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
AmString getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject getSource();
```

toString

Returns an AmString representation of the Exception.

```
AmString toString();
```

AmErrorException

An Exception of type **AmErrorException** is raised when an object experiences an error with a severity level of FAILED (CompletionCode = AMCC_FAILED).

getClassName

Returns the type of object throwing the Exception.

```
AmString getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
AmString getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject getSource();
```

toString

Returns an AmString representation of the Exception.

```
AmString toString();
```

AmWarningException

An Exception of type **AmWarningException** is raised when an object experiences an error with a severity level of WARNING (CompletionCode = AMCC_WARNING).

getClassName

Returns the type of object throwing the Exception.

```
AmString getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
AmString getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject getSource();
```

toString

Returns an AmString representation of the Exception.

```
AmString toString();
```

Part 4. The COBOL interface

Chapter 9. Using the Application Messaging

Interface in COBOL	227
Structure of the AMI	227
Using the repository	228
System default objects	228
Writing applications in COBOL	230
Opening and closing a session.	230
Sending messages	230
Using the message object	231
Sample programs	232
Receiving messages	232
Using the message object	233
Sample programs	233
Request/response messaging	234
Request	234
Response	234
Sample programs	235
File transfer	235
Publish/subscribe messaging	235
Publish	235
Subscribe	236
Sample programs	237
Using name/value elements	237
Example	239
Error handling	239
Transaction support	240
Sending group messages	240
Other considerations	240
Multithreading	240
Using MQSeries with the AMI.	240
Field limits	240
Building COBOL applications	241
COBOL applications on OS/390	241
AMI Copybooks	241
Preparing COBOL programs on OS/390	241
Running COBOL programs on OS/390	242

Chapter 10. The COBOL high-level interface

Overview of the COBOL high-level interface	244
Initialize and terminate	244
Sending messages	244
Receiving messages	244
File transfer	244
Publish/subscribe	244
Transaction support	244
Reference information for the COBOL high-level interface	246
AMHBACK (backout)	247
AMHBEGIN (begin)	248
AMHBRMS (browse message)	249
AMHCMIT (commit)	251
AMHINIT (initialize)	252
AMHPB (publish)	253
AMHRCFL (receive file)	254
AMHRCMS (receive message)	256
AMHRCPB (receive publication)	258

AMHRCRQ (receive request)	260
AMHSNFL (send file)	262
AMHSNMS (send message)	263
AMHSNRQ (send request)	264
AMHSNRS (send response)	265
AMHSB (subscribe)	266
AMHTERM (terminate)	267
AMHUN (unsubscribe)	268

Chapter 11. COBOL object interface overview

Session interface functions	270
Session management	270
Create objects	270
Get object handles	270
Delete objects	271
Transactional processing	271
Error handling	271
Message interface functions	272
Get values	272
Set values	272
Reset values	272
Read and write data	272
Publish/subscribe topics	273
Publish/subscribe filters	273
Publish/subscribe name/value elements	273
Error handling	273
Sender interface functions	274
Open and close	274
Send	274
Get values	274
Error handling	274
Receiver interface functions	275
Open and close	275
Receive and browse	275
Get values	275
Set values	275
Error handling	275
Distribution list interface functions	276
Open and close	276
Send	276
Get values	276
Error handling	276
Publisher interface functions	277
Open and close	277
Publish	277
Get values	277
Error handling	277
Subscriber interface functions	278
Open and close	278
Broker messages	278
Get values	278
Set value	278
Error handling	278
Policy interface functions	279
Get values	279
Set value	279

Error handling	279
High-level functions	280

Chapter 12. COBOL object interface reference 283

Session interface functions	284
AMSEBG (begin)	284
AMSECLEC (clear error codes)	284
AMSECL (close)	285
AMSECM (commit)	285
AMSECR (create)	286
AMSECRDL (create distribution list)	286
AMSECRMS (create message)	287
AMSECRPO (create policy)	287
AMSECRPB (create publisher)	288
AMSECRRC (create receiver)	288
AMSECRSN (create sender)	289
AMSECRSB (create subscriber)	289
AMSEDL (delete)	290
AMSEDLDL (delete distribution list)	290
AMSEDLMS (delete message)	290
AMSEDLPO (delete policy)	291
AMSEDLPB (delete publisher)	291
AMSEDLRC (delete receiver)	291
AMSEDLSN (delete sender)	292
AMSEDLNB (delete subscriber)	292
AMSEGHDL (get distribution list handle)	292
AMSEGTLE (get last error codes)	293
AMSEGHMS (get message handle)	293
AMSEGHPO (get policy handle)	294
AMSEGHPB (get publisher handle)	294
AMSEGHRC (get receiver handle)	294
AMSEGHSN (get sender handle)	295
AMSEGHSB (get subscriber handle)	295
AMSEOP (open)	296
AMSERB (rollback)	296
Message interface functions	297
AMMSADEL (add element)	298
AMMSADFI (add filter)	298
AMMSADTO (add topic)	299
AMMSCLEC (clear error codes)	299
AMMSDEEL (delete element)	299
AMMSDEFI (delete filter)	300
AMMSDENE (delete named element)	300
AMMSDETO (delete topic)	301
AMMSGELC (get element CCSID)	301
AMMSGTCC (get CCSID)	301
AMMSGTCI (get correl ID)	302
AMMSGTDL (get data length)	302
AMMSGTDO (get data offset)	302
AMMSGTEL (get element)	303
AMMSGTEC (get element count)	303
AMMSGTEN (get encoding)	304
AMMSGTFC (get filter count)	304
AMMSGTFI (get filter)	305
AMMSGTFO (get format)	305
AMMSGTGS (get group status)	306
AMMSGTLE (get last error)	306
AMMSGTMI (get message ID)	307
AMMSGTNA (get name)	307
AMMSGTNE (get named element)	308
AMMSGTNC (get named element count)	308

AMMSGTRC (get report code)	309
AMMSGTTO (get topic)	309
AMMSGTTC (get topic count)	310
AMMSGTTY (get type)	310
AMMSREBY (read bytes)	311
AMMSRS (reset)	311
AMMSSTCC (set CCSID)	312
AMMSSTCI (set correl ID)	312
AMMSSTDO (set data offset)	312
AMMSSELC (set element ccsid)	313
AMMSSTEN (set encoding)	313
AMMSSTFO (set format)	314
AMMSSTGS (set group status)	314
AMMSWRBY (write bytes)	315
Sender interface functions	316
AMSNCLEC (clear error codes)	316
AMSNCL (close)	317
AMSNGTCC (get CCSID)	317
AMSNGTEN (get encoding)	317
AMSNGTLE (get last error)	318
AMSNGTNA (get name)	318
AMSNOP (open)	319
AMSNNS (send)	319
AMSNNSFL (send file)	320
Usage notes	320
Receiver interface functions	322
AMRCBR (browse)	322
Usage notes	323
AMRCBRSE (browse selection message)	324
Usage notes	325
AMRCCLEC (clear error codes)	325
AMRCCL (close)	326
AMRCGTD (get definition type)	326
AMRCGTLE (get last error)	327
AMRCGTNA (get name)	327
AMRCGTQN (get queue name)	328
AMRCOP (open)	328
AMRCRC (receive)	329
Usage notes	329
AMRCRCFL (receive file)	330
AMRCSTQN (set queue name)	331
Distribution list interface functions	332
AMDLCLEC (clear error codes)	332
AMDLCCL (close)	332
AMDLTLE (get last error)	332
AMDLTNA (get name)	333
AMDLTSC (get sender count)	333
AMDLTSH (get sender handle)	334
AMDLOP (open)	334
AMDLSN (send)	335
AMDLSNFL (send file)	335
Usage notes	336
Publisher interface functions	337
AMPBCLEC (clear error codes)	337
AMPBCL (close)	337
AMPBGTC (get CCSID)	337
AMPBGTEN (get encoding)	338
AMPBGTLE (get last error)	338
AMPBGTNA (get name)	339
AMPBOP (open)	339
AMPBPB (publish)	340

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

Chapter 9. Using the Application Messaging Interface in COBOL

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

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

This chapter describes the following:

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

Structure of the AMI

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

Session	Contains the AMI session.
Message	Contains the message data, message ID, correlation ID, and options that are used when sending or receiving a message (most of which come from the policy definition).
Sender	This is a service that represents a destination (such as an MQSeries queue) to which messages are sent.
Receiver	This is a service that represents a source from which messages are received.
Distribution list	Contains a list of sender services to provide a list of destinations.
Publisher	Contains a sender service where the destination is a publish/subscribe broker.
Subscriber	Contains a sender service (to send subscribe and unsubscribe messages to a publish/subscribe broker) and a receiver service (to receive publications from the broker).
Policy	Defines how the message should be handled, including items such as priority, persistence, and whether it is included in a unit of work.

When using the high-level functions the objects are created automatically and (where applicable) populated with values from the repository. In some cases it might be necessary to inspect these properties after a message has been sent (for instance, the *MessageID*), or to change the value of one or more properties before sending the message (for instance, the *Format*). To satisfy these requirements, the AMI for COBOL has a lower-level object style interface in addition to the high-level procedural interface. This provides access to the objects listed above,

Structure of the AMI

with methods to *set* and *get* their properties. You can mix high-level and object-level functions in the same application.

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

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

Using the repository

You can run AMI applications with or without a repository. If you don't have a repository, you can use a system default object (see below), or create your own by specifying its name on a high-level function call. It will be created using the appropriate system provided definition (see “System provided definitions” on page 470).

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

System default objects

Table 3. System default objects

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

A set of system default objects is created at session creation time. This removes the overhead of creating the objects from applications using these defaults. The system default objects are available for use from both the high-level and object interfaces in COBOL. They are created using the system provided definitions (see “System provided definitions” on page 470).

Structure of the AMI

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

Constants representing synonyms for handles are also provided for these objects, for use from the object interface (see “Appendix B. Constants” on page 509). Note that the first parameter on a call must be a real handle; you cannot use a synonym in this case (that is why handles are not provided for all the default objects).

Writing applications in COBOL

This section gives a number of examples showing how to use the high-level interface of the AMI, with some extensions using the object interface. Equivalent operations to all high-level functions can be performed using combinations of object interface functions (see “High-level functions” on page 280).

Opening and closing a session

Before using the AMI, you must open a session. This can be done with the following high-level function (page 252):

Opening a session

```
CALL 'AMHINIT' USING SESSION-NAME, POLICY-NAME, HSESSION,  
                    COMPCODE, REASON.
```

The SESSION-NAME is optional. POLICY-NAME is the name of the policy to be used during initialization of the AMI. If it consists of a space or low value, the SYSTEM.DEFAULT.POLICY object is used. Or you can specify the constant AMSD-POL to use the default policy.

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

To close a session, you can use this high-level function (page 267):

Closing a session

```
CALL 'AMHTERM' USING HSESSION, POLICY-NAME, COMPCODE, REASON.
```

This closes and deletes all objects that were created in the session.

Sending messages

You can send a datagram (send and forget) message using the high-level AMHSNMS function (page 263). In the simplest case, all you need to specify is the session handle returned by AMHINIT, the message data, and the message length. Other parameters can be specified using the constants that represent the default message, sender service, and policy objects.

Sending a message using all the defaults

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

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

Sending a message using a specified sender service

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

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

Sending a message using a specified policy

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

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

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

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

Sending a message to a distribution list

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

Using the message object

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

Specifying a message object

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

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

Writing applications in COBOL

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

Setting an attribute in a message object

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

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

Look at “Message interface functions” on page 272 to find out what other attributes of the message object you can get and set.

After a message object has been used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see AMMSRS on page 311) and rebuild it each time.

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

Sample programs

For more details, refer to the AMTVHSND and AMTVOSND sample programs (see “Sample programs for OS/390” on page 464).

Receiving messages

Use the AMHRCMS high-level function (page 256) to receive a message to which no response is to be sent (such as a datagram). In the simplest case, all you need to specify are the session handle and a buffer for the message data. Other parameters can be specified using the constants that represent the default message, receiver service, and policy objects.

Receiving a message using all the defaults

```
CALL 'AMHRCMS' USING HSESSION, AMSD-RCV, AMSD-POL, AMSD-SND-MSG,  
                    BUFFLEN, DATALEN, DATA, AMSD-RCV-MSG,  
                    COMPCODE, REASON.
```

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

Receiving a message using a specified receiver service

```
CALL 'AMHRCMS' USING HSESSION, RECEIVER-NAME, AMSD-POL, AMSD-SND-MSG,  
                    BUFFLEN, DATALEN, DATA, AMSD-RCV-MSG,  
                    COMPCODE, REASON.
```

Writing applications in COBOL

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

Receiving a message using a specified policy

```
CALL 'AMHRCMS' USING HSESSION, AMSD-RCV, POLICY-NAME, AMSD-SND-MSG,  
                    BUFFLEN, DATALEN, DATA, AMSD-RCV-MSG,  
                    COMPCODE, REASON.
```

The policy can specify, for example:

- The wait interval
- If the message is part of a unit of work
- If the message should be code page converted
- If all the members of a group must be there before any members can be read

Using the message object

To get the attributes of a message after receiving it, you can specify your own message object name, or use the system default SYSTEM.DEFAULT.RECEIVE.MESSAGE (constant: AMSD-RCV-MSG). If a message object of that name does not exist it will be created. You can access the attributes (such as the *Encoding*) using the object interface functions:

Getting an attribute from a message object

```
CALL 'AMHRCMS' USING HSESSION, AMSD-RCV, AMSD-POL, AMSD-SND-MSG,  
                    BUFFLEN, DATALEN, DATA, RECEIVE-MSG,  
                    COMPCODE, REASON.  
  
CALL 'AMSEGHMS' USING HSESSION, RECEIVE-MSG, HMSG, COMPCODE, REASON.  
  
CALL 'AMMSGTEN' USING HMSG, ENCODING, COMPCODE, REASON.
```

If a specific message is to be selectively received using its correlation identifier, a message object must first be created and its *CorrelId* property set to the required value (using the object interface). This message object is passed as the *selection message* on the AMHRCMS call:

Using a selection message object

```
CALL 'AMSECRMS' USING HSESSION, SELECTION-MSG, HMSG, COMPCODE, REASON.  
  
CALL 'AMMSSTCI' USING HMSG, CORRELIDLLEN, CORRELID, COMPCODE, REASON.  
  
CALL 'AMHRCMS' USING HSESSION, AMSD-RCV, AMSD-POL, SELECTION-MSG,  
                    BUFFLEN, DATALEN, DATA, AMSD-RCV-MSG,  
                    COMPCODE, REASON.
```

Sample programs

For more details, refer to the AMTVHRCV and AMTVORCV sample programs (see “Sample programs for OS/390” on page 464).

Writing applications in COBOL

Request/response messaging

In the *request/response* style of messaging, a requester (or client) application sends a request message and expects to receive a message in response. The responder (or server) application receives the request message and produces the response message (or messages) which it returns to the requester application. The responder application uses information in the request message to determine how to send the response message to the requester.

In the following examples 'CLIENT' refers to the requesting application, and 'SERVER' refers to the responding application.

Request

Use the AMHSNRQ high-level function (page 264) to send a request message. This is similar to AMHSNMS, but it includes the name of the service to which the response message is to be sent. In this example the sender service (CLIENT-SENDER) is specified in addition to the receiver service (CLIENT-RECEIVER). A send message name (CLIENT-SND-MSG) is specified as well.

Sending a request message

```
CALL 'AMHSNRQ' USING HSESSION, CLIENT-SENDER, AMSD-POL, CLIENT-RECEIVER,  
                    DATALEN, DATA, CLIENT-SND-MSG, COMPCODE, REASON.
```

The AMHRCRQ high-level function (page 260) is used by the responding (or server) application to receive a request message. It is similar to AMHRCMS, but it includes the name of the sender service that will be used for sending the response message. When the message is received, the sender service is updated with the information needed for sending the response to the required destination.

Receiving a request message

```
CALL 'AMHRCRQ' USING HSESSION, SERVER-RECEIVER, AMSD-POL, BUFFLEN,  
                    DATALEN, DATA, SERVER-RCV-MSG, SERVER-SENDER,  
                    COMPCODE, REASON.
```

A policy name can be specified as well, as described in "Receiving messages" on page 232.

A receiver message name (SERVER-RCV-MSG) is specified so that the response message can refer to it. Note that, unlike AMHRCMS, this function does not have a selection message.

Response

After the requested actions have been performed, the responding application sends the response message (or messages) with the AMHSNRS function (page 265):

Sending a response message

```
CALL 'AMHSNRS' USING HSESSION, SERVER-SENDER, AMSD-POL, SERVER-RCV-MSG,  
                    DATALEN, DATA, AMSD-SND-MSG, COMPCODE, REASON.
```

The sender service for the response message (SERVER-SENDER) and the receiver message name (SERVER-RCV-MSG) are the same as those used with AMHRCRQ

(receive request). This causes the *CorrelId* and *MessageId* to be set in the response message, as requested by the flags in the request message.

Finally, the requester (or client) application uses the AMHRCMS function to receive the response message as described in “Receiving messages” on page 232. You might need to receive a specific response message (for example if three request messages have been sent, and you want to receive the response to the first request message first). In this case the sender message name from the AMHSNRQ function (CLIENT-SND-MSG) should be used as the selection message name in AMHRCMS.

Sample programs

For more details, refer to the AMTVHCLT, AMTVOCLT, AMTVHSVR, and AMTSOSVR sample programs (see “Sample programs for OS/390” on page 464).

File transfer

You can perform file transfers using the AMHSNFL and AMHRCFL high-level functions, and the AMSNSNFL, AMDLSNFL and AMRCRCFL object-level functions.

Sending a file using the high-level AMHSNFL function

```
CALL 'AMHSNFL' USING HSESSION, SENDER-NAME, POLICYNAME, OPTIONS,  
                    FILENAME-LENGTH, FILENAME, SNDMSG-NAME.
```

Receiving a file using the high-level AMHRCFL function

```
CALL 'AMHRCFL' USING HSESSION, RECEIVER-NAME, POLICY-NAME, OPTIONS,  
                    SELMSG-NAME, FILENAME-LENGTH, SNDMSG-NAME.
```

For a complete description of file transfer, refer to “File transfer” on page 21

Publish/subscribe messaging

With *publish/subscribe* messaging, *publisher* applications publish messages to *subscriber* applications using a *broker*. The messages published contain application data and one or more *topic* strings that describe the data. Subscribing applications register subscriptions informing the broker which topics they are interested in. When the broker receives a published message, it forwards the message to all subscribing applications for which a topic in the message matches a topic in the subscription.

Subscribing applications can exploit content-based publish/subscribe by passing a filter on subscribe and unsubscribe calls (see “Using MQSeries Integrator Version 2” on page 458).

For more information, refer to the *MQSeries Publish/Subscribe User's Guide*.

Publish

Use the AMHPB high-level function (page 253) to publish a message. You need to specify the name of the publisher for the publish/subscribe broker (or use the default by specifying AMSD-PUB).

Writing applications in COBOL

The topic relating to this publication and the publication data must also be specified:

Publishing a message

```
CALL 'AMHPB' USING HSESSION, PUBLISHER-NAME, AMSD-POL, RECEIVER-NAME,  
TOPICLEN, TOPIC, DATALEN, DATA, PUBLISH-MSG,  
COMPCODE, REASON.
```

The RECEIVER-NAME identifies the receiver service to which the broker will send a response message. You can also specify a policy name to change the behavior of the function (as with the AMHSNxx functions).

You can specify the publication message name PUBLISH-MSG and set or get attributes of the message object (using the object interface functions). This might include adding another topic (using AMMSADTO) before invoking AMHPB, if there are multiple topics associated with this publication.

Instead of sending the publication data using the data buffer, it can be added to the message object. Unlike the AMHSNxx functions, this gives no difference in performance with large messages. This is because, whichever method is used, the MQRFH header has to be added to the publication data before sending it (similarly the header has to be removed when the publication is received).

Subscribe

The AMHSB high-level function (page 266) is used to subscribe to a publish/subscribe broker specified by the name of a subscriber service. The receiver to which publications will be sent is included within the definition of the subscriber. The name of a receiver service to which the broker can send a response message (RECEIVER-NAME) is also specified.

Subscribing to a broker

```
CALL 'AMHSB' USING HSESSION, SUBSCRIBER-NAME, AMSD-POL, RECEIVER-NAME,  
TOPICLEN, TOPIC, 0, 0, SUBSCRIBE-MSG,  
COMPCODE, REASON.
```

A subscription for a single topic can be passed by the TOPIC parameter. You can subscribe to multiple topics by using the object interface AMMSADTO function to add topics to the SUBSCRIBE-MSG message object, before invoking AMHSB.

If the policy specifies that the *CorrelId* is to be used as part of the identity for the subscribing application, it can be added to the subscription message object with the object interface AMMSSTCI function, before invoking AMHSB.

To remove a subscription, use the AMHUN high-level function (page 268). To remove all subscriptions, you can specify a policy that has the 'Deregister All Topics' subscriber attribute.

To receive a publication from a broker, use the AMHRCPB function (page 258). For example:

Receiving a publication

```
CALL 'AMHRCPB' USING HSESSION, SUBSCRIBER-NAME, AMSD-POL, SELECTION-MSG,
                    TOPICBUFFLEN, BUFFLEN, TOPICCOUNT, TOPICLEN,
                    FIRSTTOPIC, DATALEN, DATA, RECEIVE-MSG,
                    COMPCODE, REASON.
```

You need to specify the name of the subscriber service used for the original subscription. You can also specify a policy name and a selection message name, as described in “Receiving messages” on page 232.

If there are multiple topics associated with the publication, only the first one is returned by this function. So, if TOPICCOUNT indicates that there are more topics, you have to access them from the RECEIVE-MSG message object, using the object-level AMSEGHMS (get message handle) and AMMSGTTO (get topic) functions.

Sample programs

For more details, refer to the AMTVHPUB, AMTSOPUB, AMTVHSUB, and AMTSOSUB sample programs (see “Sample programs for OS/390” on page 464).

Using name/value elements

Publish/subscribe brokers (such as MQSeries Publish/Subscribe) respond to messages that contain name/value pairs to define the commands and options to be used. The AMHPB, AMHSB, AMHUN, and AMHRCPB high-level functions provide these name/value pairs implicitly.

For less commonly used commands and options, the name/value pairs can be added to a message using an AMELEM structure. The AMTELEMV and AMTELEML copybooks define the AMELEM structure, with and without default values respectively. Here is the AMTELEMV copybook:

```
** AMELEM structure
10 AMELEM.
** Structure identifier
15 AMELEM-STRUCID PIC X(8) VALUE 'COEL '.
** Structure version number
15 AMELEM-VERSION PIC S9(9) BINARY VALUE 1.
** Reserved, must be zero
15 AMELEM-GROUP-BUFF-LEN PIC S9(9) BINARY VALUE 0.
** Reserved, must be zero
15 AMELEM-GROUP-LEN PIC S9(9) BINARY VALUE 0.
** Reserved, must be zero
15 AMELEM-GROUP-OFFSET PIC S9(9) BINARY VALUE 0.
** Name buffer length
15 AMELEM-NAME-BUFF-LEN PIC S9(9) BINARY VALUE 0.
** Name length in bytes
15 AMELEM-NAME-LEN PIC S9(9) BINARY VALUE 0.
** Name
15 AMELEM-NAME-OFFSET PIC S9(9) BINARY VALUE 0.
** Value buffer length
15 AMELEM-VALUE-BUFF-LEN PIC S9(9) BINARY VALUE 0.
** Value length in bytes
15 AMELEM-VALUE-LEN PIC S9(9) BINARY VALUE 0.
** Value
15 AMELEM-VALUE-OFFSET PIC S9(9) BINARY VALUE 0.
** Reserved, must be zero
```

Writing applications in COBOL

15	AMELEM-TYPE-BUFF-LEN	PIC S9(9) BINARY VALUE 0.
**	Reserved, must be zero	
15	AMELEM-TYPE-LEN	PIC S9(9) BINARY VALUE 0.
**	Reserved, must be zero	
15	AMELEM-TYPE-OFFSET	PIC S9(9) BINARY VALUE 0.

The offset fields in the AMELEM structure allow you to give the location of the name and value buffers relative to the start of the AMELEM structure. The offsets can be positive or negative.

Following are short descriptions of the fields and an example of how to use the AMELEM structure.

AMELEM-STRUCID

The AMELEM structure identifier (input).

AMELEM-VERSION

The version number of the AMELEM structure (input). Its value must be one.

AMELEM-GROUP-BUFF-LEN

Reserved, must be zero.

AMELEM-GROUP-LEN

Reserved, must be zero.

AMELEM-GROUP-OFFSET

Reserved, must be zero.

AMELEM-NAME-BUFF-LEN

The length of the name buffer (input). If this field is set to zero, the AMI returns the name length value (in AMELEM-NAME-LEN) but not the name value (in AMELEM-NAME-OFFSET). This is not an error.

AMELEM-NAME-LEN

The length of the name in bytes (input or output).

AMELEM-NAME-OFFSET

The name buffer (input or output).

AMELEM-VALUE-BUFF-LEN

The length of the value buffer (input).

AMELEM-VALUE-LEN

The value length in bytes (input or output).

AMELEM-VALUE-OFFSET

The value buffer (input or output).

AMELEM-TYPE-BUFF-LEN

Reserved, must be zero.

AMELEM-TYPE-LEN

Reserved, must be zero.

AMELEM-TYPE-OFFSET

Reserved, must be zero.

Example

As an example, to send a message containing a 'Request Update' command, define the command data and the AMELEM structure as follows:

```
01 OPTIONS                      PIC S9(9) BINARY VALUE ZERO.
01 AMELEM-DATA.
   10 COMMAND-NAME              PIC X(16) VALUE 'MQPSCommand'.
   10 COMMAND-VALUE            PIC X(16) VALUE 'ReqUpdate'.
   COPY AMTELEMV.
```

Set the length and offset values as follows:

```
MOVE 11 TO AMELEM-NAME-LEN.
MOVE -48 TO AMELEM-NAME-OFFSET.
MOVE 9 TO AMELEM-VALUE-LEN.
MOVE -32 TO AMELEM-VALUE-OFFSET.
```

Having set the values, create a message object (SEND-MSG) and add the element to it:

Using name/value elements

```
CALL 'AMSECRMS' USING HSESSION, SEND-MSG, HMSG, COMPCODE, REASON.

CALL 'AMMSADEL' USING HMSG, AMELEM, OPTIONS, COMPCODE, REASON.
```

You must then send the message, using AMHSNMS, to the sender service specified for the publish/subscribe broker.

If you need to use streams with MQSeries Publish/Subscribe, you must add the appropriate stream name/value element explicitly to the message object.

The message element functions can, in fact, be used to add any element to a message before issuing a publish/subscribe request. Such elements (including topics, which are specialized elements) supplement or override those added implicitly by the request, as appropriate to the individual element type.

The use of name/value elements is not restricted to publish/subscribe applications. They can be used in other applications as well.

Error handling

Each AMI COBOL function returns a completion code reflecting the success or failure (OK, warning, or error) of the request. Information indicating the reason for a warning or error is returned in a reason code.

The 'get last error' functions (such as AMSEGTLE) always reflect the last most severe error detected by an object. These functions can be used to return the completion and reason codes associated with this error. Once the error has been handled, call the 'clear error codes' functions (for instance, AMMSCLEC) to clear the error information.

All COBOL high-level functions record last error information in the session object. This information can be accessed using the session's 'get last error' call, AMSEGTLE (you need the session handle returned by AMHINIT as the first parameter of this call).

Writing applications in COBOL

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
AMGRP_LAST_MSG_IN_GROUP for the last message
```

The message status is set using **AMMSSTGS**.

For a complete description of group messages, refer to “Sending group messages” on page 26.

Other considerations

You should consider the following when writing your applications:

- Multithreading
- Using MQSeries with the AMI
- Field limits

Multithreading

Multithreading is not supported for COBOL applications running on OS/390.

Using MQSeries with the AMI

You must not mix MQSeries function calls with AMI function calls within the same process.

Field limits

When string and binary properties such as queue name, message format, and correlation ID are set, the maximum length values are determined by MQSeries, the underlying message transport. See the rules for naming MQSeries objects in the *MQSeries Application Programming Guide*.

Building COBOL applications

The Application Messaging Interface for COBOL is available only on the OS/390 operating system.

COBOL applications on OS/390

This section explains what you have to do to prepare and run your COBOL programs on the OS/390 operating system. See “Language compilers” on page 426 for compilers supported by the AMI.

AMI Copybooks

The AMI provides COBOL copybooks to assist you with the writing of your applications. The copybook AMTV contains constants and return codes. Copybooks AMTELEML and AMTELEMV contain the definition of the AMELEM data structure that is used to pass name/value element information across the AMI. AMTELEML provides a data definition without initial values; AMTELEMV provides the same definition with initial values.

These copybooks are installed in the MQSeries for OS/390 library hlq.SCSQCOBC. Use the COPY statement to include them in your program. For example:

```
WORKING STORAGE SECTION.
01  AMI-CONSTANTS.
    COPY AMTV.
```

You are recommended to use the copybook AMTELEMV to define an AMELEM structure. This provides default initial values which ensures that the *strucId* and *version* fields have valid values. If the values passed for these fields are not valid, the AMI will reject them.

Preparing COBOL programs on OS/390

COBOL programs that use the AMI must be compiled and linked edited. Programs containing CICS commands must be processed by the CICS translator before compilation. To add AMI support, include the appropriate COBOL stub (interface module) in the link edit. The AMI provides a COBOL stub for each supported environment (batch, RRS batch, or CICS), as follows:

Batch	AMTBS10
RRS batch	AMTRS10
CICS	AMTCS10
IMS	AMTIS10

Note: If you are using COBOL, you should select the NODYNAM compiler option to enable the linkage editor to resolve references to the AMI stub.

Thus the link edit JCL should specify a ‘DD’ name for the MQSeries for OS/390 hlq.SCSQLOAD library and an INCLUDE statement for the stub. For example, to link edit an AMI batch application:

```
//LKED EXEC PGM=HEWL....
....
//OBJLIB DD DSN=th1qua1.SCSQLOAD,DISP=SHR
//SYSIN DD *
  ENTRY CEESTART
  INCLUDE OBJLIB(AMTBS10)
  NAME progname(R)
/*
```

COBOL applications on OS/390

Running COBOL programs on OS/390

The AMI needs access to the MQSeries datasets SCSQLOAD and SCSQAUTH, as well as one of the language-specific datasets such as SCSQANLE. See the *MQSeries Application Programming Guide* for details of the supported languages.

For CICS operation, the library hlq.SCSQLOAD and the Language Environment[®] SCEERUN library must be included in the DFHRPL concatenation. COBOL programs using the AMI must be defined to CICS with a language code of 'Le370'.

For information about AMI tracing, see "Using trace (OS/390)" on page 489.

Chapter 10. The COBOL high-level interface

The COBOL high-level interface contains functions that cover the requirements of the majority of applications. If extra functionality is needed, COBOL object interface functions can be used in the same application as the COBOL high-level functions.

This chapter contains:

- “Overview of the COBOL high-level interface” on page 244
- “Reference information for the COBOL high-level interface” on page 246

Overview of the COBOL high-level interface

The high-level functions are listed below. Follow the page references to see the detailed descriptions of each function.

Initialize and terminate

Functions to create and open an AMI session, and to close and delete an AMI session.

AMHINIT (initialize)	page 252
AMHTERM (terminate)	page 267

Sending messages

Functions to send a datagram (send and forget) message, and to send request and response messages.

AMHSNMS (send message)	page 263
AMHSNRQ (send request)	page 264
AMHSNRS (send response)	page 265

Receiving messages

Functions to receive a message from AMHSNMS or AMHSNRS, to receive a request message from AMHSNRQ, and to browse a message.

AMHRCMS (receive message)	page 256
AMHRCRQ (receive request)	page 260
AMHBRMS (browse message)	page 249

File transfer

Functions to send message data from a file, and to receive message data sent by AMHSNFL into a file.

AMHSNFL (send file)	page 262
AMHRCFL (receive file)	page 254

Publish/subscribe

Functions to publish a message to a publish/subscribe broker, and to subscribe, unsubscribe, and receive publications.

AMHPB (publish)	page 253
AMHSB (subscribe)	page 266
AMHUN (unsubscribe)	page 268
AMHRCPB (receive publication)	page 258

Transaction support

Functions to begin, commit and back out a unit of work.

AMHBEGIN (begin)	page 248
-------------------------	----------

AMHCMIT (commit)

page 251

AMHBACK (backout)

page 247

Reference information for the COBOL high-level interface

In the following sections the high-level interface functions are listed in alphabetical order. Note that all functions return a completion code (COMPCODE) and a reason code (REASON). The completion code can take one of the following values:

AMCC-OK	Function completed successfully
AMCC-WARNING	Function completed with a warning
AMCC-FAILED	An error occurred during processing

If the completion code returns warning or failed, the reason code identifies the reason for the error or warning (see “Appendix A. Reason codes” on page 497).

Object names can be up to `AMLEN-MAX-NAME-LENGTH` characters, and are terminated by a space or by a low value (a single byte zero). If a space or low value is not found, the name will be truncated at `AMLEN-MAX-NAME-LENGTH`.

If an object name is specified as a space or low value, the relevant system default name will be used.

Most functions require the session handle to be specified. If this handle is not valid, the results are unpredictable.

AMHBACK (backout)

Function to back out a unit of work.

```
CALL 'AMHBACK' USING HSESSION, POLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.  
01 POLICY        PIC X(n).  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

HSESSION The session handle returned by AMHINIT (input).

POLICY The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMHBEGIN (begin)

Function to begin a unit of work.

```
CALL 'AMHBEGIN' USING HSESSION, POLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.  
01 POLICY        PIC X(n).  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

HSESSION The session handle returned by AMHINIT (input).

POLICY The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMHBRMS (browse message)

Function to browse a message. See the *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.
01 RECEIVER      PIC X(n) .
01 POLICY        PIC X(n) .
01 OPTIONS       PIC S9(9) BINARY.
01 BUFFLEN       PIC S9(9) BINARY.
01 DATALEN      PIC S9(9) BINARY.
01 DATA         PIC X(n) .
01 RCVMSGNAME    PIC X(n) .
01 SENDER        PIC X(n) .
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HSESSION The session handle returned by AMHINIT (input).

RECEIVER The name of a receiver service (input). If specified as a space or low value, the system default receiver name (constant: AMSD-RCV) is used.

POLICY The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.

OPTIONS Options controlling the browse operation (input). Possible values are:

```
AMBRW-NEXT
AMBRW-FIRST
AMBRW-RECEIVE-CURRENT
AMBRW-DEFAULT      (AMBRW-NEXT)
```

AMBRW-RECEIVE-CURRENT is equivalent to AMRCRC (receive) for the message under the browse cursor.

BUFFLEN The length in bytes of a buffer in which the data is returned (input).

DATALEN The length of the message data, in bytes (output). Can be specified as -1 (input).

DATA The received message data (output).

RCVMSGNAME The name of the message object for the received message (input). Properties, and message data if not returned in the DATA parameter, can be extracted from the message object using the object interface (see “Message interface functions” on page 297). The message object is implicitly reset before the browse takes place. If specified as a space or low value, the system default receive message name (constant: AMSD-RCV-MSG) is used.

SENDER The name of a special type of sender service known as a *response sender*, to which the response message will be sent (input). This sender name must not have been defined in the repository before the start of the AMI session. It is only applicable if the message type is AMMT-REQUEST.

COMPCODE Completion code (output).

COBOL high-level interface

REASON Reason code (output).

Usage notes

| You can return data in the message object or in an application buffer.

| To return the data in the message object (RCVMSGNAME), rather than the
| application message buffer, set BUFFLEN to zero and set both DATA and DATALEN as
| non_NULL (not -1).

| To return data in an application message buffer:

- | • set DATA as the address of the buffer (that is, non_NULL, not -1)
- | • set BUFFLEN to the length of the buffer

| If the value of BUFFLEN is less than the length of the message data, behavior
| depends on whether Accept Truncated Message in the policy receive attributes is
| selected. If Accept Truncated Message is selected, the data is truncated and there is
| an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not
| selected (the default), the receive fails and there is an
| AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a
| non_NULL value for DATALEN (that is, not -1).

| To return only the data length:

- | • set DATA to NULL (-1)
- | • set BUFFLEN to zero
- | • ensure that Accept Truncated Message in the policy receive attributes is not
| selected

| In this way, you can determine the required buffer size before you issue a second
| receive request to return the data.

AMHCMIT (commit)

Function to commit a unit of work.

```
CALL 'AMHCMIT' USING HSESSION, POLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.  
01 POLICY        PIC X(n).  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

HSESSION The session handle returned by AMHINIT (input).

POLICY The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMHINIT (initialize)

Function to create and open an AMI session. It returns a session handle, which is valid until the session is terminated.

```
CALL 'AMHINIT' USING SESSNAME, POLICY, HSESSION, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 SESSNAME      PIC X(n).  
01 POLICY        PIC X(n).  
01 HSESSION      PIC S9(9) BINARY.  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

SESSNAME	An optional name that can be used to identify the application (input).
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
HSESSION	The session handle (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHPB (publish)

Function to publish a message to a publish/subscribe broker.

```
CALL 'AMHPB' USING HSESSION, PUBLISHER, POLICY, RESPNAME,
                  TOPICLEN, TOPIC, DATALEN, DATA, MSGNAME,
                  COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.
01 PUBLISHER     PIC X(n) .
01 POLICY        PIC X(n) .
01 RESPNAME     PIC X(n) .
01 TOPICLEN     PIC S9(9) BINARY.
01 TOPIC        PIC X(n) .
01 DATALEN     PIC S9(9) BINARY.
01 DATA        PIC X(n) .
01 MSGNAME      PIC X(n) .
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
PUBLISHER	The name of a publisher service (input). If specified as a space or low value, the system default publisher name (constant: AMSD-PUB) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
RESPNAME	The name of the receiver service to which the response to this publish request will be sent (input). If specified as a space or low value, no response will be sent. This parameter is mandatory if the policy specifies implicit publisher registration (the default).
TOPICLEN	The length of the topic for this publication, in bytes (input).
TOPIC	The topic for this publication (input).
DATALEN	The length of the publication data in bytes (input). A value of zero indicates that any publication data has been added to the message object (MSGNAME) using the object interface (see “Message interface functions” on page 297).
DATA	The publication data, if DATALEN is non-zero (input).
MSGNAME	The name of a message object that contains the header for the publication message (input). If DATALEN is zero, the message object also holds any publication data. If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHRCFL (receive file)

Function to receive message data sent by AMHSNFL into a file.

```
CALL 'AMHRCFL' USING HSESSION, RECEIVERNAME, POLICYNAME,
                    OPTIONS, SELMSGNAME, DIRNAMELEN,
                    DIRNAME, FILENAMELEN, FILENAME,
                    RCVMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.
01 RECEIVERNAME  PIC X(n).
01 POLICYNAME    PIC X(n).
01 OPTIONS       PIC S9(9) BINARY.
01 SELMSGNAME    PIC X(n).
01 DIRNAMELEN    PIC S9(9) BINARY.
01 DIRNAME       PIC X(n).
01 FILENAMELEN   PIC S9(9) BINARY.
01 FILENAME      PIC X(n).
01 RCVMSGNAME    PIC X(n).
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

- HSESSION** The session handle returned by AMHINIT (input).
- RECEIVERNAME** The name of a receiver service (input). If specified as a space or low value, the system default receiver name (constant: AMSD-RCV) is used.
- POLICYNAME** The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
- OPTIONS** Reserved, must be specified as zero.
- SELMSGNAME** Optional selection message object used to specify information (such as a *CorrelId*) needed to select the required message (input).
- DIRNAMELEN** Reserved, must be specified as zero (input).
- DIRNAME** Reserved.
- FILENAMELEN** The length of the file name in bytes (input).
- FILENAME** The name of the file into which the transferred data is to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If blank then the AMI will use the name of the originating file (including any directory prefix) exactly as it was supplied on the send file call. Note that the original file name may not be appropriate for use by the receiver, either because a path name included in the file name is not applicable to the receiving system, or because the sending and receiving systems use different file naming conventions.
- RCVMSGNAME** The name of the message object to be used to receive the file (output). This parameter is updated with the message properties (for example, the Message ID). If the message is not from a file, rcvMsgName receives the message data. If specified as a blank or low value, the system default receive message name (constant AMSD-RCV-MSG) is used.

Property information and message data can be extracted from the message object using the object interface (see “Message interface functions” on page 297). The message object is reset implicitly before the receive takes place.

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 <i>CorrelId</i>) needed to select the required message (input).
BUFFLEN	The length in bytes of a buffer in which the data is returned (input). Can be specified as -1.
DATALEN	The length of the message data, in bytes (output). Can be specified as -1 (input).
DATA	The received message data (output).
RCVMSGNAME	The name of the message object for the received message (output). If specified as a space or low value, the system default receive message name (constant: AMSD-RCV-MSG) is used. Properties, and message data if not returned in the DATA parameter, can be extracted from the message object using the object interface (see "Message interface functions" on page 297). The message object is implicitly reset before the receive takes place.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

- | You can return data in the message object or in an application buffer.
- |
- | To return the data in the message object (RCVMSGNAME), rather than the
- | application message buffer, set BUFFLEN to zero and set both DATA and DATALEN as
- | non_NULL (not -1).
- |
- | To return data in an application message buffer:
- |
- | • set DATA as the address of the buffer (that is, non_NULL, not -1)
- |
- | • set BUFFLEN to the length of the buffer

COBOL high-level interface

If the value of `BUFFLEN` is less than the length of the message data, behavior depends on whether `Accept Truncated Message` in the policy receive attributes is selected. If `Accept Truncated Message` is selected, the data is truncated and there is an `AMRC_MSG_TRUNCATED` warning. If `Accept Truncated Message` is not selected (the default), the receive fails and there is an `AMRC_RECEIVE_BUFF_LEN_ERR` error. To return the data length, set a non-NULL value for `DATALEN` (that is, not `-1`).

To return only the data length without removing the message from the queue:

- set `DATA` to NULL (`-1`)
- set `BUFFLEN` to zero
- ensure that `Accept Truncated Message` in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

To remove the message from the queue and discard it:

- set `DATA` or `DATALEN` to a non-NULL value (that is, not `-1`)
- set `BUFFLEN` to zero
- ensure that `Accept Truncated Message` in the policy receive attributes is selected

The message will be discarded with an `AMRC_MSG_TRUNCATED` warning.

If `AMRC_RECEIVE_BUFF_LEN_ERR` is returned, the message length value is returned in `DATALEN` (if it is non-NULL, that is, not `-1`), even though the completion code is `MQCC_FAILED`.

Note that if `DATA` is NULL (`-1`) and `BUFFLEN` is not zero, there is always an `AMRC_RECEIVE_BUFF_LEN_ERR` error.

AMHRCPB (receive publication)

Function to receive a publication from a publish/subscribe broker.

```
CALL 'AMHRCPB' USING HSESSION, SUBSCRIBER, POLICY, SELMSGNAME,
                    TOPICBUFFLEN, BUFFLEN, TOPICCOUNT, TOPICLEN,
                    FIRSTTOPIC, DATALEN, DATA, RCVMSGNAME,
                    COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.
01 SUBSCRIBER    PIC X(n).
01 POLICY        PIC X(n).
01 SELMSGNAME    PIC X(n).
01 TOPICBUFFLEN  PIC S9(9) BINARY.
01 BUFFLEN       PIC S9(9) BINARY.
01 TOPICCOUNT   PIC S9(9) BINARY.
01 TOPICLEN      PIC S9(9) BINARY.
01 FIRSTTOPIC    PIC X(n).
01 DATALEN      PIC S9(9) BINARY.
01 DATA         PIC X(n).
01 RCVMSGNAME    PIC X(n).
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SUBSCRIBER	The name of a subscriber service (input). If specified as a space or low value, the system default subscriber name (constant: AMSD-SUB) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
SELMSGNAME	Optional selection message object used to specify information (such as a <i>CorrelId</i>) needed to select the required message (input).
TOPICBUFFLEN	The length in bytes of a buffer in which the topic is returned (input).
BUFFLEN	The length in bytes of a buffer in which the publication data is returned (input).
TOPICCOUNT	The number of topics in the message (output).
TOPICLEN	The length in bytes of the first topic (output).
FIRSTTOPIC	The first topic (output). Topics can be extracted from the message object (RCVMSGNAME) using the object interface (see “Message interface functions” on page 297).
DATALEN	The length in bytes of the publication data (output).
DATA	The publication data (output). Data can be extracted from the message object (RCVMSGNAME) using the object interface (see “Message interface functions” on page 297).
RCVMSGNAME	The name of a message object for the received message (input). If specified as a space or low value, the system default message name (constant: AMSD-RCV-MSG) is used. The publication message properties and data update this message object, in addition to being returned in the parameters above. The message object is implicitly reset before the receive takes place.
COMPCODE	Completion code (output).

REASON Reason code (output).

Usage notes

We recommend that, when using AMHRCPB, you always have data conversion enabled in the specified policy. If data conversion is not enabled, AMHRCPB will fail if the local CCSID and/or encoding values differ from those on the platform from which the publication was sent.

If data conversion is enabled by the specified policy, and a selection message is specified, the conversion is performed using the target encoding and coded character set identifier (CCSID) values designated in the selection message. (The selection message is specified in the SELMSGNAME parameter).

If a selection message is not specified, the platform encoding and Queue Manager CCSID values are used as defaults for the conversion.

If a normal message that is not a publication message is received by the specified subscriber, AMHRCPB behaves the same as AMHRCMS.

AMHRCRQ (receive request)

Function to receive a request message.

```
CALL 'AMHRCRQ' USING HSESSION, RECEIVER, POLICY, BUFFLEN, DATALEN,  
DATA, RCVMSGNAME, SENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.  
01 RECEIVER      PIC X(n).  
01 POLICY        PIC X(n).  
01 BUFFLEN       PIC S9(9) BINARY.  
01 DATALEN      PIC S9(9) BINARY.  
01 DATA         PIC X(n).  
01 RCVMSGNAME    PIC X(n).  
01 SENDER        PIC X(n).  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
RECEIVER	The name of a receiver service (input). If specified as a space or low value, the system default receiver name (constant: AMSD-RCV) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
BUFFLEN	The length in bytes of a buffer in which the data is returned (input).
DATALEN	The length of the message data, in bytes (output). Can be specified as -1 (input).
DATA	The received message data (output).
RCVMSGNAME	The name of the message object for the received message (output). If specified as NULL, the system default receiver service (constant: AMSD-RCV-MSG) is used. Header information, and message data if not returned in the DATA parameter, can be extracted from the message object using the object interface (see “Message interface functions” on page 297). The message object is implicitly reset before the receive takes place.
SENDER	The name of a special type of sender service known as a <i>response sender</i> , to which the response message will be sent (output). This sender name must not be defined in the repository. If specified as a space or low value, the system default response sender service (constant: AMSD-RSP-SND) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

The following notes contain details about use of the AMHRCRQ function.

Data conversion

If data conversion is enabled by the specified policy, and a selection message is specified, the conversion is performed using the target encoding and coded character set identifier (CCSID) values designated in the selection message. (These target values are specified in the SELMSGNAME parameter).

If a selection message is not specified, the platform encoding and Queue Manager CCSID values are used as defaults for the conversion.

Use of the buffLen parameter

You can return data in the message object or in an application buffer.

To return the data in the message object (RCVMSGNAME), rather than the application message buffer, set BUFFLEN to zero and set both DATA and DATALEN as non_NULL (not -1).

To return data in an application message buffer:

- set DATA as the address of the buffer (that is, non_NULL, not -1)
- set BUFFLEN to the length of the buffer

If the value of BUFFLEN is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a non_NULL value for DATALEN (that is, not -1).

To return only the data length without removing the message from the queue:

- set DATA to NULL (-1)
- set BUFFLEN to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

To remove the message from the queue and discard it:

- set DATA or DATALEN to a non_NULL value (that is, not -1)
- set BUFFLEN to zero
- ensure that Accept Truncated Message in the policy receive attributes is selected

The message will be discarded with an AMRC_MSG_TRUNCATED warning.

If AMRC_RECEIVE_BUFF_LEN_ERR is returned, the message length value is returned in DATALEN (if it is non_NULL, that is, not -1), even though the completion code is MQCC_FAILED.

Note that if DATA is NULL (-1) and BUFFLEN is not zero, there is always an AMRC_RECEIVE_BUFF_LEN_ERR error.

AMHSNFL (send file)

Function to send data from a file.

```
CALL 'AMHSNFL' USING HSESSION, SENDERNAME, POLICYNAME,  
                    OPTIONS, DIRNAMELEN, DIRNAME,  
                    FILENAMELEN, FILENAME,  
                    SNDMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.  
01 SENDERNAME    PIC X(n).  
01 POLICYNAME    PIC X(n).  
01 OPTIONS       PIC S9(9) BINARY.  
01 DIRNAMELEN    PIC S9(9) BINARY.  
01 DIRNAME       PIC X(n).  
01 FILENAMELEN   PIC S9(9) BINARY.  
01 FILENAME      PIC X(n).  
01 SNDMSGNAME    PIC X(n).  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SENDERNAME	The name of a sender service (input). If specified as a space or low value, the system default sender name (constant: AMSD-SND) is used.
POLICYNAME	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
OPTIONS	Reserved, must be specified as zero.
DIRNAMELEN	Reserved, must be specified as zero (input).
DIRNAME	Reserved.
FILENAMELEN	The length of the file name in bytes (input).
FILENAME	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with a receive file call (see "AMHRCFL (receive file)" on page 254 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
SNDMSGNAME	The name of the message object to be used to send the file (input). This can be used to specify the Correlation ID for example. The Correlation ID can be set from the message object using the object interface (see "Message interface functions" on page 297). If SNDMSGNAME is specified as a space or low value, the system default send message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

The message object is implicitly reset by this call.

The system default object is used when you set SNDMSGNAME as a space or low value.

AMHSNMS (send message)

Function to send a datagram (send and forget) message.

```
CALL 'AMHSNMS' USING HSESSION, SENDER, POLICY, DATALEN, DATA,
                    SNDMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.
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 297).

DATA The message data, if DATALEN is non-zero (input).

SNDMSGNAME The name of a message object for the message being sent (input). If DATALEN is zero, the message object also holds any message data. If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMHSNRQ (send request)

Function to send a request message.

```
CALL 'AMHSNRQ' USING HSESSION, SENDER, POLICY, RESPNAME, DATALEN,  
                    DATA, SNDMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.  
01 SENDER        PIC X(n).  
01 POLICY        PIC X(n).  
01 RESPNAME     PIC X(n).  
01 DATALEN     PIC S9(9) BINARY.  
01 DATA        PIC X(n).  
01 SNDMSGNAME   PIC X(n).  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SENDER	The name of a sender service (input). If specified as a space or low value, the system default sender name (constant: AMSD-SND) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
RESPNAME	The name of the receiver service to which the response to this send request will be sent (input). See AMHRCRQ (receive request).
DATALEN	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (SNDMSGNAME) using the object interface (see "Message interface functions" on page 297).
DATA	The message data, if DATALEN is non-zero (input).
SNDMSGNAME	The name of a message object for the message being sent (input). If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHSNRS (send response)

Function to send a response to a request message.

```
CALL 'AMHSNRS' USING HSESSION, SENDER, POLICY, RCVMSGNAME, DATALEN,
                    DATA, SNDMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.
01 SENDER        PIC X(n).
01 POLICY        PIC X(n).
01 RCVMSGNAME    PIC X(n).
01 DATALEN     PIC S9(9) BINARY.
01 DATA         PIC X(n).
01 SNDMSGNAME    PIC X(n).
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SENDER	The name of the sender service (input). It must be set to the SENDER specified for the AMHRCRQ receive request.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
RCVMSGNAME	The name of the received message that this message is a response to (input). It must be set to the RCVMSGNAME specified for the AMHRCRQ receive request.
DATALEN	The length of the message data in bytes (input). A value of zero indicates that any message data has been added to the message object (SNDMSGNAME) using the object interface (see “Message interface functions” on page 297).
DATA	The message data, if DATALEN is non-zero (input).
SNDMSGNAME	The name of a message object for the message being sent (input). If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMHSB (subscribe)

Function to register a subscription with a publish/subscribe broker.

Publications matching the subscription are sent to the receiver service associated with the subscriber. By default, this has the same name as the subscriber service, with the addition of the suffix '.RECEIVER'.

Subscribing applications can exploit content based publish/subscribe by passing a filter on the AMHSUB call.

```
CALL 'AMHSB' USING HSESSION, SUBSCRIBER, POLICY, RESPNAME,
                  TOPICLEN, TOPIC, FILTERLEN, FILTER,
                  SUBMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.
01 SUBSCRIBER    PIC X(n).
01 POLICY        PIC X(n).
01 RESPNAME      PIC X(n).
01 TOPICLEN      PIC S9(9) BINARY.
01 TOPIC         PIC X(n).
01 FILTERLEN     PIC S9(9) BINARY.
01 FILTER        PIC X(n).
01 SUBMSGNAME    PIC X(n).
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

- HSESSION** The session handle returned by AMHINIT (input).
- SUBSCRIBER** The name of a subscriber service (input). If specified as a space or low value, the system default subscriber name (constant: AMSD-SUB) is used.
- POLICY** The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
- RESPNAME** The name of the receiver service to which the response to this subscribe request will be sent (input). If specified as a space or low value, no response is sent.
- This is not the service to which publications will be sent by the broker; they are sent to the receiver service associated with the subscriber (see above).
- TOPICLEN** The length of the topic for this subscription, in bytes (input).
- TOPIC** The topic for this subscription (input). Publications that match this topic, including wildcards, will be sent to the subscriber. Multiple topics can be specified in the message object (SUBMSGNAME) using the object interface (see "Message interface functions" on page 297).
- FILTERLEN** The length in bytes of the filter (input).
- FILTER** The filter to be added (input). The syntax of the filter string is described in the *MQSeries Integrator Version 2.0 Programming Guide*.
- SUBMSGNAME** The name of a message object for the subscribe message (input). If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
- COMPCODE** Completion code (output).
- REASON** Reason 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).

REASON Reason code (output).

AMHUN (unsubscribe)

Function to remove a subscription from a publish/subscribe broker.

```
CALL 'AMHUN' USING HSESSION, SUBSCRIBER, POLICY, RESPNAME,
                  TOPICLEN, TOPIC, FILTERLEN, FILTER,
                  UNSUBMSGNAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESSION      PIC S9(9) BINARY.
01 SUBSCRIBER    PIC X(n).
01 POLICY        PIC X(n).
01 RESPNAME      PIC X(n).
01 TOPICLEN      PIC S9(9) BINARY.
01 TOPIC         PIC X(n).
01 FILTERLEN     PIC S9(9) BINARY.
01 FILTER        PIC X(n).
01 UNSUBMSGNAME  PIC X(n).
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HSESSION	The session handle returned by AMHINIT (input).
SUBSCRIBER	The name of a subscriber service (input). If specified as a space or low value, the system default subscriber name (constant: AMSD-SUB) is used.
POLICY	The name of a policy (input). If specified as a space or low value, the system default policy name (constant: AMSD-POL) is used.
RESPNAME	The name of the receiver service to which the response to this unsubscribe request will be sent (input).
TOPICLEN	The length of the topic, in bytes (input).
TOPIC	The topic that identifies the subscription which is to be removed (input). Multiple topics can be specified in the message object (UNSUBMSGNAME) using the object interface (see “Message interface functions” on page 297). To deregister all topics, a policy providing this option must be specified (this is not the default policy). Otherwise, to remove a previous subscription the topic information specified must match that specified on the relevant AMHSB subscribe request.
FILTERLEN	The length in bytes of the filter (input). A value of AMLEN_NULL_TERM specifies that the string is null terminated.
FILTER	The filter that identifies the subscription to be removed (input). The syntax of the filter string is described in the <i>MQSeries Integrator Version 2.0 Programming Guide</i>
UNSUBMSGNAME	The name of a message object for the unsubscribe message (input). If specified as a space or low value, the system default message name (constant: AMSD-SND-MSG) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

To successfully remove a previous subscription, you must ensure that the topic, filter, and subscriber queue information exactly matches that used on the original subscribe request.

Chapter 11. COBOL object interface overview

This chapter contains an overview of the structure of the COBOL object interface. Use it to find out what functions are available in this interface.

The object interface provides sets of interface functions for each of the following objects:

Session	page 270
Message	page 272
Sender	page 274
Receiver	page 275
Distribution list	page 276
Publisher	page 277
Subscriber	page 278
Policy	page 279

These interface functions are invoked as necessary by the high-level functions. They are made available to the application programmer through this object-style interface to provide additional function where needed. An application program can mix high-level functions and object-interface functions as required.

Details of the interface functions for each object are given in the following pages. Follow the page references to see the detailed descriptions of each function.

Details of the object interface functions used by each high-level function are given on page 280.

Session interface functions

The session object creates and manages all other objects, and provides the scope for a unit of work.

Session management

Functions to create, open, close, and delete a session object.

AMSECR (create)	page 286
AMSEOP (open)	page 296
AMSECL (close)	page 285
AMSEDL (delete)	page 290

Create objects

Functions to create message, sender, receiver, distribution list, publisher, subscriber, and policy objects. Handles to these objects are returned by these functions.

AMSECRMS (create message)	page 287
AMSECRSN (create sender)	page 289
AMSECRRC (create receiver)	page 288
AMSECRDL (create distribution list)	page 286
AMSECRPB (create publisher)	page 288
AMSECRSB (create subscriber)	page 289
AMSECRPO (create policy)	page 287

Get object handles

Functions to get the handles for a message, sender, receiver, distribution list, publisher, subscriber, and policy objects with a specified name (needed if the objects were created implicitly by the high-level interface).

AMSEGHMS (get message handle)	page 293
AMSEGHSN (get sender handle)	page 295
AMSEGHRC (get receiver handle)	page 294
AMSEGHDL (get distribution list handle)	page 292
AMSEGHPB (get publisher handle)	page 294
AMSEGHSB (get subscriber handle)	page 295
AMSEGHPO (get policy handle)	page 294

Delete objects

Functions to delete message, sender, receiver, distribution list, publisher, subscriber, and policy objects.

AMSEDLMS (delete message) page 290

AMSEDLN (delete sender) page 292

AMSEDLRC (delete receiver) page 291

AMSELDL (delete distribution list)
page 290

AMSEDLPB (delete publisher)
page 291

AMSEDLNB (delete subscriber)
page 292

AMSEDLPO (delete policy) page 291

Transactional processing

Functions to begin, commit, and rollback a unit of work.

AMSEBG (begin) page 284

AMSECM (commit) page 285

AMSERB (rollback) page 296

Error handling

Functions to clear the error codes, and return the completion and reason codes for the last error associated with the session object.

AMSECLC (clear error codes)
page 284

AMSEGTLE (get last error codes)
page 293

Message interface functions

A message object encapsulates an MQSeries message descriptor (MQMD) structure. It also contains the message data if this is not passed as a separate parameter.

Get values

Functions to get the coded character set ID, correlation ID, encoding, format, group status, message ID, name, report code, and type of the message object.

AMMSGTCC (get CCSID)	page 301
AMMSGTCI (get correl ID)	page 302
AMMSGELC (get element CCSID)	page 301
AMMSGTEN (get encoding)	page 304
AMMSGTFO (get format)	page 305
AMMSGTGS (get group status)	page 306
AMMSGTMI (get message ID)	page 307
AMMSGTNA (get name)	page 307
AMMSGTRC (get report code)	page 309
AMMSGTTY (get type)	page 310

Set values

Functions to set the coded character set ID, correlation ID, encoding, format, and group status of the message object.

AMMSSTCC (set CCSID)	page 312
AMMSSTCI (set correl ID)	page 312
AMMSSELC (set element CCSID)	page 313
AMMSSTEN (set encoding)	page 313
AMMSSTFO (set format)	page 314
AMMSSTGS (set group status)	page 314

Reset values

Function to reset the message object to the state it had when first created.

AMMSRS (reset)	page 311
-----------------------	----------

Read and write data

Functions to get the length of the data, get and set the data offset, and read or write byte data to or from the message object at the current offset.

AMMSGTDL (get data length)	page 302
-----------------------------------	----------

AMMSGTDO (get data offset)	page 302
AMMSSTDO (set data offset)	page 312
AMMSREBY (read bytes)	page 311
AMMSWRBY (write bytes)	page 315

Publish/subscribe topics

Functions to manipulate the topics in a publish/subscribe message.

AMMSADTO (add topic)	page 299
AMMSDETO (delete topic)	page 301
AMMSGTTO (get topic)	page 309
AMMSGTTC (get topic count)	page 310

Publish/subscribe filters

Functions to manipulate the filters in a publish/subscribe message.

AMMSADFI (add filter)	page 298
AMMSDEFI (delete filter)	page 300
AMMSGTFI (get filter)	page 305
AMMSGTFC (get filter count)	page 304

Publish/subscribe name/value elements

Functions to manipulate the name/value elements in a publish/subscribe message.

AMMSADEL (add element)	page 298
AMMSDEEL (delete element)	page 299
AMMSGTEL (get element)	page 303
AMMSGTEC (get element count)	page 303
AMMSDENE (delete named element)	page 300
AMMSGTNE (get named element)	page 308
AMMSGTNC (get named element count)	page 308

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the message.

AMMSCLEC (clear error codes)	page 299
AMMSGTLE (get last error)	page 306

Sender interface functions

A sender object encapsulates an MQSeries object descriptor (MQOD) structure for sending a message.

Open and close

Functions to open and close the sender service.

AMSNOP (open) page 319

AMSNCL (close) page 317

Send

Function to send a message.

AMSNSN (send) page 319

AMSNSNFL(send file) page 320

Get values

Functions to get the coded character set ID, encoding, and name of the sender service.

AMSNGTCC (get CCSID) page 317

AMSNGTEN (get encoding) page 317

AMSNGTNA (get name) page 318

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the sender service.

AMSNCLEC (clear error codes)
page 316

AMSNGTLE (get last error) page 318

Receiver interface functions

A receiver object encapsulates an MQSeries object descriptor (MQOD) structure for receiving a message.

Open and close

Functions to open and close the receiver service.

AMRCOP (open) page 328

AMRCCL (close) page 326

Receive and browse

Functions to receive or browse a message.

AMRCRC (receive) page 329

AMRCRCFL (receive file) page 330

AMRCBR (browse) page 322

AMRCBRSE (browse selection message)
page 324

Get values

Functions to get the definition type, name, and queue name of the receiver service.

AMRCGTD (get definition type)
page 326

AMRCGTNA (get name) page 327

AMRCGTQN (get queue name)
page 328

Set values

Function to set the queue name of the receiver service.

AMRCSTQN (set queue name)
page 331

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the receiver service.

AMRCCLEC (clear error codes)
page 325

AMRCGTLE (get last error) page 327

Distribution list interface functions

A distribution list object encapsulates a list of sender services.

Open and close

Functions to open and close the distribution list service.

AMDLOP (open) page 334

AMDLCCL (close) page 332

Send

Function to send a message to the distribution list.

AMDLSN (send) page 335

AMDLSNFL (send file) page 335

Get values

Functions to get the name of the distribution list service, a count of the sender services in the list, and a sender service handle.

AMD LGTNA (get name) page 333

AMD LGTSC (get sender count)
page 333

AMD LGTSH (get sender handle)
page 334

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the distribution list.

AMDLCLEC (clear error codes)
page 332

AMD LGTLE (get last error) page 332

Publisher interface functions

A publisher object encapsulates a sender service. It provides support for publishing messages to a publish/subscribe broker.

Open and close

Functions to open and close the publisher service.

AMPBOP (open) page 339

AMPBCL (close) page 337

Publish

Function to publish a message.

AMPBPB (publish) page 340

Get values

Functions to get the coded character set ID, encoding, and name of the publisher service.

AMPBGTCC (get CCSID) page 337

AMPBGTEN (get encoding) page 338

AMPBGTNA (get name) page 339

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the publisher.

AMPBCLEC (clear error codes)
page 337

AMPBGTLE (get last error) page 338

Subscriber interface functions

A subscriber object encapsulates both a sender service and a receiver service. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

Open and close

Functions to open and close the subscriber service.

AMSBOP (open)	page 345
AMSBCL (close)	page 341

Broker messages

Functions to subscribe to a broker, remove a subscription, and receive publications from the broker.

AMSBBSB (subscribe)	page 346
AMSBUN (unsubscribe)	page 347
AMSBRC (receive)	page 345

Get values

Functions to get the coded character set ID, definition type, encoding, name, and queue name of the subscriber service.

AMSBGTCC (get CCSID)	page 342
AMSBGTDT (get definition type)	page 342
AMSBGTEN (get encoding)	page 343
AMSBGTNA (get name)	page 344
AMSBGTQN (get queue name)	page 344

Set value

Function to set the queue name of the subscriber service.

AMSBSTQN (set queue name)	page 346
----------------------------------	----------

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the receiver.

AMSBCLEC (clear error codes)	page 341
AMSBGTLE (get last error)	page 343

Policy interface functions

A policy object encapsulates details of how the message is handled (such as priority, persistence, and whether it is included in a unit of work).

Get values

Functions to get the name of the policy, and the wait time set in the policy.

AMPOGTNA (get name) page 349

AMPOGTWT (get wait time) page 349

Set value

Function to set the wait time for a receive using the policy.

AMPOSTWT (set wait time) page 350

Error handling

Functions to clear the error codes, and return the completion and reason codes from the last error associated with the policy.

AMPOCLEC (clear error codes)
page 348

AMPOGTLE (get last error) page 348

High-level functions

Each high-level function described in “Chapter 10. The COBOL high-level interface” on page 243 calls a number of the object interface functions, as shown below.

Table 4. Object interface calls used by the high-level functions

High-level function	Equivalent object interface calls
AMHBACK (backout)	AMSECRPO / AMSEGHPO AMSERB
AMHBEGIN (begin)	AMSECRPO / AMSEGHPO AMSEBG
AMHBRMS (browse message)	AMSECRRC / AMSEGHRC AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMRCBRSE
AMHCMIT (commit)	AMSECRPO / AMSEGHPO AMSECM
AMHINIT (initialize)	AMSECR AMSEOP
AMHTERM (terminate)	AMSECL AMSEDL
AMHSNMS (send message) AMHSNRQ (send request) AMHSNRS (send response)	AMSECRSN / AMSEGHSN AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMSNSN
AMHRCMS (receive message) AMHRCRQ (receive request)	AMSECRRC / AMSEGHRC AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMRCRC
AMHSNFL (send file)	AMSECRSN / AMSEGHSN AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMSNSNFL
AMHRCFL (receive file)	AMSECRRC / AMSEGHRC AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMRCRCFL
AMHPB (publish)	AMSECRPB / AMSEGHPB AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMPBPB
AMHSB (subscribe)	AMSECRSB / AMSEGHSB AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMSBBS
AMHUN (unsubscribe)	AMSECRSB / AMSEGHSB AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMSBUN
AMHRCPB (receive publication)	AMSECRSB / AMSEGHSB AMSECRPO / AMSEGHPO AMSECRMS / AMSEGHMS AMSBRC

COBOL object interface overview

If an object already exists, the appropriate call to get its handle is used instead of calling the create function again. For example, if the policy object exists, AMSEGHPO (get policy handle) is used instead of AMSECRPO (create policy).

Chapter 12. COBOL object interface reference

In the following sections the COBOL object interface functions are listed by the object they refer to:

Session	page 284
Message	page 297
Sender	page 316
Receiver	page 322
Distribution list	page 332
Publisher	page 337
Subscriber	page 341
Policy	page 348

Within each section the functions are listed in alphabetical order.

Note that all functions return a completion code (COMPCODE) and a reason code (REASON). The completion code can take one of the following values:

AMCC-OK	Function completed successfully
AMCC-WARNING	Function completed with a warning
AMCC-FAILED	An error occurred during processing

If the completion code returns warning or failed, the reason code identifies the reason for the error or warning (see "Appendix A. Reason codes" on page 497).

Most functions require a handle to the object they reference. If this handle is not valid, the results are unpredictable.

Session interface functions

A *session* object provides the scope for a unit of work and creates and manages all other objects, including at least one connection object. Each (MQSeries) connection object encapsulates a single MQSeries queue manager connection. The session object definition specifying the required queue manager connection can be provided by a repository policy definition and the local host file, or the local host file only which by default will name a single local queue manager with no repository. (Under CICS, there can be only one queue manager connected to a given CICS system, so in this case the local host file is irrelevant.) The session, when deleted, is responsible for releasing memory by closing and deleting all other objects that it manages.

Note that you should not mix MQSeries MQCONN or MQDISC requests on the same thread as AMI calls, otherwise premature disconnection might occur.

AMSEBG (begin)

Begins a unit of work, allowing an AMI application to take advantage of the resource coordination provided in MQSeries. The unit of work can subsequently be committed by AMSECM, or backed out by AMSERB. It should be used only when MQSeries is the transaction coordinator. If an external transaction coordinator (for example, CICS or Tuxedo) is being used, the API of the external coordinator should be used instead.

```
CALL 'AMSEBG' USING HSESS, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.
01 HPOLICY        PIC S9(9) BINARY.
01 COMPCODE       PIC S9(9) BINARY.
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).

REASON Reason code (output).

COBOL session interface

AMSECR (create)

Creates the session and system default objects. AMSECR returns the handle of the session object. This must be specified by other session function calls.

```
CALL 'AMSECR' USING NAME, HSESS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 NAME          PIC X(n).
01 HSESS         PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

NAME An optional session name that can be used to identify the application from which a message is sent (input).

HSESS The handle of the session object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSECRDL (create distribution list)

Creates a distribution list object. A distribution list handle is returned.

```
CALL 'AMSECRDL' USING HSESS, NAME, HDISTLIST, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS         PIC S9(9) BINARY.
01 NAME          PIC X(n).
01 HDISTLIST     PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the distribution list (input). This must match the name of a distribution list defined in the repository.

HDISTLIST The handle of the distribution list object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSECRMS (create message)

Creates a message object. A message handle is returned.

```
CALL 'AMSECRMS' USING HSESS, NAME, HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.
01 NAME           PIC X(n).
01 HMSG           PIC S9(9) BINARY.
01 COMPCODE       PIC S9(9) BINARY.
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the message (input). This can be any name that is meaningful to the application. It is specified so that this message object can be used with the high-level interface.

HMSG The handle of the message object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSECRPO (create policy)

Creates a policy object. A policy handle is returned.

```
CALL 'AMSECRPO' USING HSESS, NAME, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.
01 NAME           PIC X(n).
01 HPOLICY        PIC S9(9) BINARY.
01 COMPCODE       PIC S9(9) BINARY.
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the policy (input). If it matches a policy defined in the repository, the policy will be created using the repository definition, otherwise it will be created with default values.

If a repository is being used and the named policy is not found in the repository, a completion code of AMCC-WARNING is returned with a reason code of AMRC-POLICY-NOT-IN-REPOS.

HPOLICY The handle of the policy object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL session interface

AMSECRPB (create publisher)

Creates a publisher object. A publisher handle is returned.

```
CALL 'AMSECRPB' USING HSESS, NAME, HPUBLISHER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.  
01 NAME           PIC X(n).  
01 HPUBLISHER     PIC S9(9) BINARY.  
01 COMPCODE       PIC S9(9) BINARY.  
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the publisher (input). If it matches a publisher defined in the repository, the publisher will be created using the repository definition, otherwise it will be created with default values (that is, with a sender service name that matches the publisher name).

If a repository is being used and the named publisher is not found in the repository, a completion code of AMCC-WARNING is returned with a reason code of AMRC-PUBLISHER-NOT-IN-REPOS.

HPUBLISHER The handle of the publisher object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSECRRC (create receiver)

Creates a receiver service object. A receiver handle is returned.

```
CALL 'AMSECRRC' USING HSESS, NAME, HRECEIVER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.  
01 NAME           PIC X(n).  
01 HRECEIVER      PIC S9(9) BINARY.  
01 COMPCODE       PIC S9(9) BINARY.  
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the receiver service (input). If it matches a receiver defined in the repository, the receiver will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the receiver name).

If a repository is being used and the named receiver is not found in the repository, a completion code of AMCC-WARNING is returned with a reason code of AMRC-RECEIVER-NOT-IN-REPOS.

HRECEIVER The handle of the receiver object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSECRSN (create sender)

Creates a sender service object. A sender handle is returned.

```
CALL 'AMSECRSN' USING HSESS, NAME, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.
01 NAME           PIC X(n).
01 HSENDER        PIC S9(9) BINARY.
01 COMPCODE       PIC S9(9) BINARY.
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the sender service (input). If it matches a sender defined in the repository, the sender will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the sender name).

If a repository is being used and the named sender is not found in the repository, a completion code of AMCC-WARNING is returned with a reason code of AMRC-SENDER-NOT-IN-REPOS.

HSENDER The handle of the sender object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSECRSB (create subscriber)

Creates a subscriber object. A subscriber handle is returned.

```
CALL 'AMSECRSB' USING HSESS, NAME, HSUBSCRIBER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.
01 NAME           PIC X(n).
01 HSUBSCRIBER    PIC S9(9) BINARY.
01 COMPCODE       PIC S9(9) BINARY.
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the subscriber (input). If it matches a subscriber defined in the repository, the subscriber will be created using the repository definition, otherwise it will be created with default values (that is, with a sender service name that matches the subscriber name, and a receiver service name that is the same with the addition of the suffix '.RECEIVER').

If a repository is being used and the named subscriber is not found in the repository, a completion code of AMCC-WARNING is returned with a reason code of AMRC-SUBSCRIBER-NOT-IN-REPOS.

HSUBSCRIBER The handle of the subscriber object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

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).

HMSG The message handle returned by AMSECRMS (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSEDLPO (delete policy)

Deletes a policy object.

```
CALL 'AMSEDLPO' USING HSESS, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.
01 HPOLICY        PIC S9(9) BINARY.
01 COMPCODE       PIC S9(9) BINARY.
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).
HPOLICY The policy handle returned by AMSECRPO (input).
COMPCODE Completion code (output).
REASON Reason code (output).

AMSEDLPB (delete publisher)

Deletes a publisher object, and performs an implicit close if the publisher is open.

```
CALL 'AMSEDLPB' USING HSESS, HPUBLISHER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.
01 HPUBLISHER     PIC S9(9) BINARY.
01 COMPCODE       PIC S9(9) BINARY.
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).
HPUBLISHER The publisher handle returned by AMSECRPB (input).
COMPCODE Completion code (output).
REASON Reason code (output).

AMSEDLRC (delete receiver)

Deletes a receiver object, and performs an implicit close if the receiver is open.

```
CALL 'AMSEDLRC' USING HSESS, HRECEIVER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.
01 HRECEIVER      PIC S9(9) BINARY.
01 COMPCODE       PIC S9(9) BINARY.
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).
HRECEIVER The receiver handle returned by AMSECRRC (input).
COMPCODE Completion code (output).
REASON Reason code (output).

COBOL session interface

AMSEDLSN (delete sender)

Deletes a sender object, and performs an implicit close if the sender is open.

```
CALL 'AMSEDLSN' USING HSESS, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.  
01 HSENDER        PIC S9(9) BINARY.  
01 COMPCODE       PIC S9(9) BINARY.  
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

HSENDER The sender handle returned by AMSECRSN (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSEDLSB (delete subscriber)

Deletes a subscriber object, and performs an implicit close if the subscriber is open.

```
CALL 'AMSEDLSB' USING HSESS, HSUBSCRIBER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.  
01 HSUBSCRIBER    PIC S9(9) BINARY.  
01 COMPCODE       PIC S9(9) BINARY.  
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

HSUBSCRIBER The subscriber handle returned by AMSECRSB (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSEGHDL (get distribution list handle)

Returns the handle of the distribution list object with the specified name.

```
CALL 'AMSEGHDL' USING HSESS, NAME, HDISTLIST, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.  
01 NAME           PIC X(n).  
01 HDISTLIST      PIC S9(9) BINARY.  
01 COMPCODE       PIC S9(9) BINARY.  
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the distribution list (input).

HDISTLIST The handle of the distribution list object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSEGTLE (get last error codes)

Gets the information (completion and reason codes) from the last error for the session.

```
CALL 'AMSEGTLE' USING HSESS, BUFFLEN, STRINGLEN, ERRORTXT,
                    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 ERRORTXT       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).

ERRORTXT 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).

REASON Reason code (output).

COBOL session interface

AMSEGHPO (get policy handle)

Returns the handle of the policy object with the specified name.

```
CALL 'AMSEGHPO' USING HSESS, NAME, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.  
01 NAME           PIC X(n).  
01 HPOLICY        PIC S9(9) BINARY.  
01 COMPCODE       PIC S9(9) BINARY.  
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the policy (input).

HPOLICY The handle of the policy object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSEGHPB (get publisher handle)

Returns the handle of the publisher object with the specified name.

```
CALL 'AMSEGHPB' USING HSESS, NAME, HPUBLISHER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.  
01 NAME           PIC X(n).  
01 HPUBLISHER     PIC S9(9) BINARY.  
01 COMPCODE       PIC S9(9) BINARY.  
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the publisher (input).

HPUBLISHER The handle of the publisher object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSEGHRC (get receiver handle)

Returns the handle of the receiver service object with the specified name.

```
CALL 'AMSEGHRC' USING HSESS, NAME, HRECEIVER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.  
01 NAME           PIC X(n).  
01 HRECEIVER      PIC S9(9) BINARY.  
01 COMPCODE       PIC S9(9) BINARY.  
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the receiver (input).

HRECEIVER The handle of the receiver object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSEGHSN (get sender handle)

Returns the handle of the sender service object with the specified name.

```
CALL 'AMSEGHSN' USING HSESS, NAME, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.
01 NAME           PIC X(n).
01 HSENDER        PIC S9(9) BINARY.
01 COMPCODE       PIC S9(9) BINARY.
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the sender (input).

HSENDER The handle of the sender object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSEGHSB (get subscriber handle)

Returns the handle of the subscriber object with the specified name.

```
CALL 'AMSEGHSB' USING HSESS, NAME, HSUBSCRIBER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.
01 NAME           PIC X(n).
01 HSUBSCRIBER    PIC S9(9) BINARY.
01 COMPCODE       PIC S9(9) BINARY.
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

NAME The name of the subscriber (input).

HSUBSCRIBER The handle of the subscriber object (output).

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL session interface

AMSEOP (open)

Opens the session object using the specified policy options. The policy, together with the local host file, provides the connection definition that enables the connection object to be created. The specified library is loaded and initialized. (Because client connections are not supported on OS/390, programs running on OS/390 must use a local queue manager). The connection to the underlying message transport (MQSeries) is then opened.

```
CALL 'AMSEOP' USING HSESS, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.  
01 HPOLICY        PIC S9(9) BINARY.  
01 COMPCODE       PIC S9(9) BINARY.  
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMSERB (rollback)

Rolls back a unit of work.

```
CALL 'AMSERB' USING HSESS, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSESS          PIC S9(9) BINARY.  
01 HPOLICY        PIC S9(9) BINARY.  
01 COMPCODE       PIC S9(9) BINARY.  
01 REASON         PIC S9(9) BINARY.
```

HSESS The session handle returned by AMSECR (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

Message interface functions

A *message* object encapsulates an MQSeries message descriptor (MQMD), and name/value elements such as the topic data for publish/subscribe messages. It can also contain the message data, or this can be passed as a separate parameter.

A name/value element in a message object is held in an AMELEM structure. See “Using name/value elements” on page 237 for details.

The initial state of the message object is:

CCSID	default queue manager CCSID
CORRELATIONID	all zeros
DATALENGTH	zero
DATAOFFSET	zero
ELEMENTCOUNT	zero
ENCODING	AMENC-NATIVE
FORMAT	AMFMT-STRING
GROUPSTATUS	AMGRP-MSG-NOT-IN-GROUP
TOPICCOUNT	zero

When a message object is used to send a message, it will not normally be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see AMMSRS on page 311) and rebuild it each time.

Note that the following calls are only valid after a session has been opened with an AMSEOP call or after you have explicitly set the element CCSID with an AMMSSELC call:

AMMSADEL (add element)	page 298
AMMSDEEL (delete element)	page 299
AMMSGTEL (get element)	page 303
AMMSGTEC (get element count)	page 303
AMMSDENE (delete named element)	page 300
AMMSGTNE (get named element)	page 308
AMMSGTNC (get named element count)	page 308
AMMSADTO (add topic)	page 299
AMMSDETO (delete topic)	page 301
AMMSGTTO (get topic)	page 309
AMMSGTTC (get topic count)	page 310

COBOL message interface

AMMSADEL (add element)

Adds a name/value element to a message (such as a publish/subscribe message).

```
CALL 'AMMSADEL' USING HMSG, AMELEM, OPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 AMELEM.  
   COPY AMTELEMV.  
01 OPTIONS      PIC S9(9) BINARY.  
01 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 <i>MQSeries Integrator Version 2.0 Programming Guide</i> .
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMMSADTO (add topic)

Adds a topic to a publish/subscribe message.

```
CALL 'AMMSADTO' USING HMSG, TOPICLEN, TOPIC, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 TOPICLEN     PIC S9(9) BINARY.
01 TOPIC        PIC X(n).
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

TOPICLEN The length in bytes of the topic (input).

TOPIC The topic to be added (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSCLEC (clear error codes)

Clears the error codes in the message object.

```
CALL 'AMMSCLEC' USING HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSDEEL (delete element)

Deletes an element with the specified index from a message (such as a publish/subscribe message). Indexing is within all elements of the message, and might include topics or filters (which are specialized elements).

```
CALL 'AMMSDEEL' USING HMSG, ELEMINDEX, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 ELEMINDEX     PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

ELEMINDEX The index of the required element in the message, starting from zero (input). On completion, elements with higher ELEMINDEX values than that specified will have their index value reduced by one.

Use AMMSGTEC to get the number of elements in the message.

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL message interface

AMMSDEFI (delete filter)

Deletes a filter from a subscribe or unsubscribe message at the specified index. Indexing is within all filters.

```
CALL 'AMMSDEFI' USING HMSG, FILTERINDEX, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 FILTERINDEX  PIC S9(9) BINARY.  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

FILTERINDEX The index of the required filter in the message, starting from zero (input). **AMMSGTFI** gets the number of filters in the message.

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSDENE (delete named element)

Deletes a named element from a message (such as a publish/subscribe message), at the specified index. Indexing is within all elements that share the same name.

```
CALL 'AMMSDENE' USING HMSG, NAMEINDEX, NAMELEN, NAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 NAMEINDEX     PIC S9(9) BINARY.  
01 NAMELEN       PIC S9(9) BINARY.  
01 NAME          PIC X(n).  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

NAMEINDEX The index of the required named element in the message (input). Specifying an index of zero deletes the *first* element with the specified name. On completion, elements with higher **NAMEINDEX** values than that specified will have their index value reduced by one.

Use **AMMSGTNC** to get the number of elements in the message with the specified name.

NAMELEN The length of the element name, in bytes (input).

NAME The name of the element to be deleted (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSDETO (delete topic)

Deletes a topic from a publish/subscribe message, at the specified index. Indexing is within all topics in the message.

```
CALL 'AMMSDETO' USING HMSG, TOPICINDEX, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 TOPICINDEX    PIC S9(9) BINARY.
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

TOPICINDEX The index of the required topic in the message, starting from zero (input). On completion, topics with higher TOPICINDEX values than that specified will have their index value reduced by one.

Use AMMSGTTC to get the number of topics in the message.

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGELC (get element CCSID)

Gets the message element CCSID. This is the coded character set identifier used for passing message element data (including topic and filter data) to or from an application.

```
CALL 'AMMSGELCC' USING HMSG, ELEMENTCCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 ELEMENTCCSID  PIC S9(9) BINARY.
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

ELEMENTCCSID The element coded character set identifier (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTCC (get CCSID)

Gets the coded character set identifier of the message.

```
CALL 'AMMSGTCC' USING HMSG, CCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 CCSID         PIC S9(9) BINARY.
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

CCSID The coded character set identifier (output).

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL message interface

AMMSGTCI (get correl ID)

Gets the correlation identifier of the message.

```
CALL 'AMMSGTCI' USING HMSG, BUFFLEN, CORRELIDLEN, CORRELID,  
                    COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 BUFFLEN      PIC S9(9) BINARY.  
01 CORRELIDLEN  PIC S9(9) BINARY.  
01 CORRELID     PIC X(n).  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

BUFFLEN The length in bytes of a buffer in which the correlation identifier is returned (input).

CORRELIDLEN The length of the correlation identifier, in bytes (output).

CORRELID The correlation identifier (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTDL (get data length)

Gets the length of the message data in the message object.

```
CALL 'AMMSGTDL' USING HMSG, LENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 LENGTH        PIC S9(9) BINARY.  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

LENGTH The length of the message data, in bytes (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTDO (get data offset)

Gets the current offset in the message data for reading or writing data bytes.

```
CALL 'AMMSGTDO' USING HMSG, OFFSET, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 OFFSET        PIC S9(9) BINARY.  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

OFFSET The byte offset in the message data (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTEL (get element)

Gets an element from a message.

```
CALL 'AMMSGTEL' USING HMSG, ELEMINDEX, ELEM, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 ELEMINDEX     PIC S9(9) BINARY.
01 ELEM.
   COPY AMTELEMV.
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

ELEMINDEX The index of the required element in the message, starting from zero (input). Use AMMSGTEC to get the number of elements in the message.

ELEM The selected element in the message (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTEC (get element count)

Gets the total number of elements in a message.

```
CALL 'AMMSGTEC' USING HMSG, COUNT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 COUNT         PIC S9(9) BINARY.
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

COUNT The number of elements in the message (output).

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL message interface

AMMSGTEN (get encoding)

Gets the value used to encode numeric data types for the message.

```
CALL 'AMMSGTEN' USING HMSG, ENCODING, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 ENCODING      PIC S9(9) BINARY.  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

ENCODING The encoding of the message (output). The following values can be returned:

```
AMENC-NATIVE  
AMENC-NORMAL  
AMENC-NORMAL-FLOAT-390  
AMENC-REVERSED  
AMENC-REVERSED-FLOAT-390  
AMENC-UNDEFINED
```

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTFC (get filter count)

Gets the total number of filters in a publish/subscribe message.

```
CALL 'AMMSGTFC' USING HMSG, COUNT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 COUNT         PIC S9(9) BINARY.  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

COUNT The number of filters (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTFI (get filter)

Get a filter from a publish/subscribe message at the specified index. Indexing is within all filters.

```
CALL 'AMMSGTFI' USING HMSG, INDEX, BUFFLEN, FILTERLEN,
                    FILTER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 INDEX         PIC S9(9) BINARY.
01 BUFFLEN      PIC S9(9) BINARY.
01 FILTERLEN    PIC S9(9) BINARY.
01 FILTER       PIC X(N),
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

INDEX The index of the required filter in the message (input). Specifying an index of zero returns the first filter. AMMSGTFC gets the number of filters in the message.

BUFFLEN The length in bytes of a buffer in which the filter is returned (input).

FILTERLEN The length of the filter, in bytes (output).

FILTER The filter (output)

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTFO (get format)

Gets the format of the message.

```
CALL 'AMMSGTFO' USING HMSG, BUFFLEN, FORMATLEN, FORMAT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 BUFFLEN      PIC S9(9) BINARY.
01 FORMATLEN    PIC S9(9) BINARY.
01 FORMAT       PIC X(n).
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

BUFFLEN The length in bytes of a buffer in which the format is returned (input).

FORMATLEN The length of the format, in bytes (output).

FORMAT The format of the message (output). The values that can be returned include the following:
 AMFMT-NONE
 AMFMT-STRING
 AMFMT-RF-HEADER

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL message interface

AMMSGTGS (get group status)

Gets the group status of the message. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group.

```
CALL 'AMMSGTGS' USING HMSG, STATUS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 STATUS        PIC S9(9) BINARY.  
01 COMPCODE      PIC S9(9) BINARY.  
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

STATUS The group status (output). It can take one of the following values:

```
AMGRP-MSG-NOT-IN-GROUP  
AMGRP-FIRST-MSG-IN-GROUP  
AMGRP-MIDDLE-MSG-IN-GROUP  
AMGRP-LAST-MSG-IN-GROUP  
AMGRP-ONLY-MSG-IN-GROUP
```

Alternatively, bitwise tests can be performed using the constants:

```
AMGF-IN-GROUP  
AMGF-FIRST  
AMGF-LAST
```

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTLE (get last error)

Gets the information (completion and reason codes) from the last error for the message object.

```
CALL 'AMMSGTLE' USING HSESS, BUFFLEN, STRINGLEN, ERRORTXT,  
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 ERRORTXT       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).

ERRORTXT 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.
```

HMSG The message handle returned by AMSECRMS (input).

BUFFLEN The length in bytes of a buffer in which the message identifier is returned (input).

MSGIDLEN The length of the message identifier, in bytes (output).

MSGID The message identifier (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTNA (get name)

Gets the name of the message object.

```
CALL 'AMMSGTNA' USING HMSG, BUFFLEN, NAMELEN, NAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 BUFFLEN       PIC S9(9) BINARY.
01 NAMELEN       PIC S9(9) BINARY.
01 NAME          PIC X(n).
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

BUFFLEN The length in bytes of a buffer in which the name is returned (input).

NAMELEN The length of the name, in bytes (output).

NAME The message object name (output).

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL message interface

AMMSGTNE (get named element)

Gets a named element from a message (such as a publish/subscribe message).

```
CALL 'AMMSGTNE' USING HMSG, NAMEINDEX, NAMELEN, NAME, ELEM  
                    COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 NAMEINDEX    PIC S9(9) BINARY.  
01 NAMELEN      PIC S9(9) BINARY.  
01 NAME         PIC X(n).  
01 ELEM.  
   COPY AMTELEMV.  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

NAMEINDEX The index of the required named element in the message (input).
Specifying an index of zero returns the first element with the specified name.

Use AMMSGTNC to get the number of elements in the message with the specified name.

NAMELEN The length of the element name, in bytes (input).

NAME The element name (input).

ELEM The selected named element in the message (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTNC (get named element count)

Gets the number of elements in a message with a specified name.

```
CALL 'AMMSGTNC' USING HMSG, NAMELEN, NAME, COUNT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 NAMELEN      PIC S9(9) BINARY.  
01 NAME         PIC X(n).  
01 COUNT        PIC S9(9) BINARY.  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

NAMELEN The length of the element name, in bytes (input).

NAME The specified element name (input).

COUNT The number of elements in the message with the specified name (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTRC (get report code)

Gets the feedback code from a message of type AMMT-REPORT. If the message type is not AMMT-REPORT, error code AMRC-MSG-TYPE-NOT-REPORT will be returned.

```
CALL 'AMMSGTRC' USING HMSG, REPORTCODE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 REPORTCODE    PIC S9(9) BINARY.
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

REPORTCODE The feedback code (output). The following values can be returned:

```
AMFB-EXPIRATION
AMFB-COA
AMFB-COD
AMFB-ERROR
```

Error code AMRC_MSG_TYPE_NOT_REPORT may be issued.

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTTO (get topic)

Gets a topic from a publish/subscribe message, at the specified index. Indexing is within all topics.

```
CALL 'AMMSGTTO' USING HMSG, TOPICINDEX, BUFFLEN, TOPICLEN, TOPIC,
                    COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 TOPICINDEX    PIC S9(9) BINARY.
01 BUFFLEN       PIC S9(9) BINARY.
01 TOPICLEN      PIC S9(9) BINARY.
01 TOPIC         PIC X(n).
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

TOPICINDEX The index of the required topic in the message (input). Specifying an index of zero returns the first topic.

Use AMMSGTTC to get the number of topics in the message.

BUFFLEN The length in bytes of a buffer in which the topic is returned (input). If BUFFLEN is specified as zero, only the topic length is returned (in TOPICLEN), not the topic itself.

TOPICLEN The length of the topic, in bytes (output).

TOPIC The topic (output).

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL message interface

AMMSGTTC (get topic count)

Gets the total number of topics in a publish/subscribe message.

```
CALL 'AMMSGTTC' USING HMSG, COUNT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 COUNT        PIC S9(9) BINARY.  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

COUNT The number of topics (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSGTTY (get type)

Gets the type from a message.

```
CALL 'AMMSGTTY' USING HMSG, TYPE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 TYPE          PIC S9(9) BINARY.  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

TYPE The message type (output). The following values can be returned:

```
AMMT-DATAGRAM  
AMMT-REQUEST  
AMMT-REPLY  
AMMT-REPORT
```

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSREBY (read bytes)

Reads up to the specified number of data bytes from the message object, starting at the current data offset. The data offset must be positioned before the end of the data for the read to be successful (see “AMMSSTDO (set data offset)” on page 312). AMMSREBY will advance the data offset by the number of bytes read, leaving the offset immediately after the last byte read.

```
CALL 'AMMSREBY' USING HMSG, READLEN, DATALEN, DATA, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 READLEN      PIC S9(9) BINARY.
01 DATALEN     PIC S9(9) BINARY.
01 DATA        PIC X(n).
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

READLEN The maximum number of bytes to be read (input). The data buffer specified by DATA must be at least this size. The number of bytes returned is the minimum of READLEN and the number of bytes between the data offset and the end of the data.

DATALEN The number of bytes read (output).

DATA The read data (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSRS (reset)

Resets the message object to its initial state (see page 297).

```
CALL 'AMMSRS' USING HMSG, OPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 OPTIONS       PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

OPTIONS Reserved, must be specified as zero (input).

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL message interface

AMMSSTCC (set CCSID)

Sets the coded character set identifier of the message.

```
CALL 'AMMSSTCC' USING HMSG, CCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 CCSID         PIC S9(9) BINARY.  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

CCSID The coded character set identifier (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSSTCI (set correl ID)

Sets the correlation identifier of the message.

```
CALL 'AMMSSTCI' USING HMSG, CORRELIDLEN, CORRELID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
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).

AMMSSTDO (set data offset)

Sets the data offset for reading or writing byte data. If the data offset is greater than the current data length, it is valid to write data into the message at that offset, but an attempt to read data will result in an error. See "AMMSREBY (read bytes)" on page 311 and "AMMSWRBY (write bytes)" on page 315.

```
CALL 'AMMSSTDO' USING HMSG, OFFSET, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 OFFSET        PIC S9(9) BINARY.  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

OFFSET The offset in bytes (input). Set an offset of zero to read or write from the start of the data.

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSSELC (set element ccsid)

This specifies the character set to be used for subsequent element message data (including topic and filter data) passed to or returned from the application. Existing elements in the message are unmodified (but will be returned in this character set). The default value of element CCSID is the queue manager CCSID.

```
CALL 'AMMSSELC' USING HMSG, ELEMENTCCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 ELEMENTCCSID PIC S9(9) BINARY.
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

ELEMENTCCSID The element coded character set identifier (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSSTEN (set encoding)

Sets the encoding of the data in the message.

```
CALL 'AMMSSTEN' USING HMSG, ENCODING, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 ENCODING      PIC S9(9) BINARY.
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

ENCODING The encoding of the message (input). It can take one of the following values:

```
AMENC-NATIVE
AMENC-NORMAL
AMENC-NORMAL-FLOAT-390
AMENC-REVERSED
AMENC-REVERSED-FLOAT-390
AMENC-UNDEFINED
```

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL message interface

AMMSSTFO (set format)

Sets the format of the message.

```
CALL 'AMMSSTFO' USING HMSG, FORMATLEN, FORMAT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 FORMATLEN    PIC S9(9) BINARY.  
01 FORMAT       PIC X(n).  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

FORMATLEN The length of the format, in bytes (input).

FORMAT The format of the message (input). It can take one of the following values, or an application defined string:

```
AMFMT-NONE  
AMFMT-STRING  
AMFMT-RF-HEADER
```

If set to AMFMT-NONE, the default format for the sender will be used (if available).

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSSTGS (set group status)

Sets the group status of the message. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group. Once you start sending messages in a group, you must complete the group before sending any messages that are not in the group.

If you specify AMGRP-MIDDLE-MSG-IN-GROUP or AMGRP-LAST-MSG-IN-GROUP without specifying AMGRP-FIRST-MSG-IN-GROUP, the behavior is the same as for AMGRP-FIRST-MSG-IN-GROUP and AMGRP-ONLY-MSG-IN-GROUP respectively.

If you specify AMGRP-FIRST-MSG-IN-GROUP out of sequence, the behavior is the same as for AMGRP-MIDDLE-MSG-IN-GROUP.

```
CALL 'AMMSSTGS' USING HMSG, STATUS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.  
01 STATUS        PIC S9(9) BINARY.  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

STATUS The group status (input). It can take one of the following values:

```
AMGRP-MSG-NOT-IN-GROUP  
AMGRP-FIRST-MSG-IN-GROUP  
AMGRP-MIDDLE-MSG-IN-GROUP  
AMGRP-LAST-MSG-IN-GROUP  
AMGRP-ONLY-MSG-IN-GROUP
```

COMPCODE Completion code (output).

REASON Reason code (output).

AMMSWRBY (write bytes)

Writes the specified number of data bytes into the message object, starting at the current data offset. See “AMMSSTDO (set data offset)” on page 312.

If the data offset is not at the end of the data, existing data is overwritten. If the data offset is set beyond the current data length, the message data between the data length and the data offset is undefined. This feature enables applications to construct messages in a non-sequential manner, but care must be taken to ensure that a message is completely filled with data before it is sent.

AMMSWRBY will advance the data offset by the number of bytes written, leaving it immediately after the last byte written.

```
CALL 'AMMSWRBY' USING HMSG, WRITELEN, BYTEDATA, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HMSG          PIC S9(9) BINARY.
01 WRITELEN     PIC S9(9) BINARY.
01 BYTEDATA     PIC X(n).
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HMSG The message handle returned by AMSECRMS (input).

WRITELEN The number of bytes to be written (input).

BYTEDATA The data bytes (input).

COMPCODE Completion code (output).

REASON Reason code (output).

Sender interface functions

A *sender* object encapsulates an MQSeries object descriptor (MQOD) structure. This represents an MQSeries queue on a local or remote queue manager. An open sender service is always associated with an open connection object (such as a queue manager connection). Support is also included for dynamic sender services (those that encapsulate model queues). The required sender service object definitions can be provided from a repository, or created without a repository definition by defaulting to the existing queue objects on the local queue manager.

The high-level functions AMHSNMS (send message), AMHSNRQ (send request), and AMHSNRS (send response) call these interface functions as required to open the sender service and send a message. Additional calls are provided here to give the application program extra functionality.

A sender service object must be created before it can be opened. This is done implicitly using the high-level functions, or the AMSECRSN (create sender) session interface functions.

A *response* sender service is a special type of sender service used for sending a response to a request message. It must be created using the default definition, and not a definition stored in a repository (see “Services and policies” on page 469). Once created, it must not be opened until used in its correct context as a response sender when receiving a request message with AMRCRC (receive) or AMHRCRQ (receive request). When opened, its queue and queue manager properties are modified to reflect the *ReplyTo* destination specified in the message being received. When first used in this context, the sender service becomes a response sender service.

AMSNCLEC (clear error codes)

Clears the error codes in the sender object.

```
CALL 'AMSNCLEC' USING HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSENDER      PIC S9(9) BINARY.  
01 COMPCODE    PIC S9(9) BINARY.  
01 REASON      PIC S9(9) BINARY.
```

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).

ENCODING The encoding (output).

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL sender interface

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, ERRORTXT,  
                    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 ERRORTXT    PIC X(n).  
01 REASON2     PIC S9(9) BINARY.  
01 COMPCODE    PIC S9(9) BINARY.  
01 REASON      PIC S9(9) BINARY.
```

HSENDER The sender handle returned by AMSECRSN (input).

BUFFLEN Reserved, must be zero (input).

STRINGLEN Reserved (output).

ERRORTXT Reserved (output).

REASON2 A secondary reason code (output). If REASON indicates AMRC-TRANSPORT-WARNING or AMRC-TRANSPORT-ERR, REASON2 gives an MQSeries reason code.

COMPCODE Completion code (output).

REASON Reason code (output). A value of AMRC-SERVICE-HANDLE-ERR indicates that the AMSNGTLE function call has itself detected an error and failed.

AMSNGTNA (get name)

Gets the name of the sender service.

```
CALL 'AMSNGTNA' USING HSENDER, BUFFLEN, NAMELEN, NAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSENDER      PIC S9(9) BINARY.  
01 BUFFLEN     PIC S9(9) BINARY.  
01 NAMELEN     PIC S9(9) BINARY.  
01 NAME        PIC X(n).  
01 COMPCODE    PIC S9(9) BINARY.  
01 REASON      PIC S9(9) BINARY.
```

HSENDER The sender handle returned by AMSECRSN (input).

BUFFLEN The length in bytes of a buffer in which the name is returned (input).

NAMELEN The length of the name, in bytes (output).

NAME The name of the sender service (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSNOP (open)

Opens the sender service.

```
CALL 'AMSNOP' USING HSENDER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSENDER      PIC S9(9) BINARY.
01 HPOLICY      PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HSENDER The sender handle returned by AMSECRSN (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMSNSN (send)

Sends a message to the destination specified by the sender service. If the sender service is not open, it will be opened (if this action is specified in the policy options).

The message data can be passed in the message object, or as a separate parameter (this means that the data is not copied into the message object before the message is sent, which might improve performance, especially if the message data is large).

```
CALL 'AMSNSN' USING HSENDER, HPOLICY, HRECEIVER, HRCVMSG, DATALEN, DATA,
                    HSNDMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSENDER      PIC S9(9) BINARY.
01 HPOLICY      PIC S9(9) BINARY.
01 HRECEIVER    PIC S9(9) BINARY.
01 HRCVMSG      PIC S9(9) BINARY.
01 DATALEN     PIC S9(9) BINARY.
01 DATA        PIC X(n).
01 HSNDMSG      PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HSENDER The sender handle returned by AMSECRSN (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

HRECEIVER The handle of the receiver service to which the response to this message should be sent, if the message being sent is a request message (input). Specify as AMH-NULL-HANDLE if no response is required.

HRCVMSG The handle of a received message that is being responded to, if this is a response message (input). Specify as AMH-NULL-HANDLE if this is not a response message.

DATALEN The length of the message data, in bytes (input). If specified as zero, any message data will be passed in the message object (HSNDMSG).

COBOL sender interface

DATA	The message data, if DATALEN is non-zero (input).
HSNDMSG	The handle of a message object that specifies the properties of the message being sent (input). If DATALEN is zero, it can also contain the message data. If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-SND-MSG-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMSNSNFL (send file)

Sends data from a file.

```
CALL 'AMSNSNFL' USING HSENDER, HPOLICY, OPTIONS, DIRNAMELEN,  
DIRNAME, FILENAMELEN, FILENAME, HSNDMSG,  
COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSENDER      PIC S9(9) BINARY.  
01 HPOLICY     PIC S9(9) BINARY.  
01 OPTIONS     PIC S9(9) BINARY.  
01 DIRNAMELEN  PIC S9(9) BINARY.  
01 DIRNAME     PIC X(n).  
01 FILENAMELEN PIC S9(9) BINARY.  
01 FILENAME    PIC X(n).  
01 HSNDMSG     PIC S9(9) BINARY.  
01 COMPCODE    PIC S9(9) BINARY.  
01 REASON      PIC S9(9) BINARY.
```

HSENDER	The sender handle returned by AMSECRSN (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
OPTIONS	A reserved field that must be specified as zero.
DIRNAMELEN	A reserved field that must be specified as zero (input).
DIRNAME	A reserved field.
FILENAMELEN	The length of the file name in bytes (input).
FILENAME	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with a receive file call (see "AMRCRCFL (receive file)" on page 330 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
HSNDMSG	The handle of a message object that specifies the properties of the message being sent (input). If specified as AMN-NULL-HANDLE, the system default send message (constant: AMN-SND-MSG-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

If, in your application, you have previously used a message object, referenced by either handle or name, to send or receive data (including AMI elements or topics),

COBOL sender interface

you will need to explicitly call AMMSRS (reset message) before re-using the object for sending a file. This applies even if you use the system default message object handle (constant: AMSD-SND-MSG-HANDLE).

Receiver interface functions

A *receiver* object encapsulates an MQSeries object descriptor (MQOD) structure. This represents a local MQSeries queue. An open receiver service is always associated with an open connection object, such as a queue manager connection. Support is also included for dynamic receiver services (that encapsulate model queues). The required receiver service object definitions can be provided from a repository or can be created automatically from the set of existing queue objects available on the local queue manager.

There is a definition type associated with each receiver service:

```
AMDT-UNDEFINED
AMDT-TEMP-DYNAMIC
AMDT-DYNAMIC
AMDT-PREDEFINED
```

A receiver service created from a repository definition will be initially of type AMDT-PREDEFINED or AMDT-DYNAMIC. When opened, its definition type might change from AMDT-DYNAMIC to AMDT-TEMP-DYNAMIC according to the properties of its underlying queue object.

A receiver service created with default values (that is, without a repository definition) will have its definition type set to AMDT-UNDEFINED until it is opened. When opened, this will become AMDT-DYNAMIC, AMDT-TEMP-DYNAMIC, or AMDT-PREDEFINED, according to the properties of its underlying queue object.

AMRCBR (browse)

Browses a message. See the *MQSeries Application Programming Guide* for a full description of the browse options.

```
CALL 'AMRCBR' USING HRECEIVER, HPOLICY, OPTIONS, BUFFLEN, DATALEN, DATA
                    HRCVMSG, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HRECEIVER    PIC S9(9) BINARY.
01 HPOLICY      PIC S9(9) BINARY.
01 OPTIONS      PIC S9(9) BINARY.
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 AMSECRRR (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

OPTIONS Options controlling the browse operation (input). Possible values are:

```
AMBRW-NEXT
AMBRW-FIRST
AMBRW-RECEIVE-CURRENT
AMBRW-DEFAULT (AMBRW-NEXT)
```

AMBRW-RECEIVE-CURRENT is equivalent to AMRCRC for the message under the browse cursor.

BUFFLEN	The length in bytes of a buffer in which the data is returned (input).
DATALEN	The length of the message data, in bytes (output). This can be set to -1 (input).
DATA	The received message data (output).
HRCVMSG	The handle of the message object for the received message (output).
HSENDER	The handle of the response sender service that the response message must be sent to, if this is a request message (output). This sender service must be created without a repository definition (that is, it must not exist before the AMI session is started), and must be used exclusively for sending a response. Its definition type must be AMDT-UNDEFINED (it will be set to AMDT-RESPONSE by this call). Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

You can return data in the message object or in an application buffer.

To return the data in the message object (HRCVMSG), rather than the application message buffer, set BUFFLEN to zero and set both DATA and DATALEN as non_NULL (not -1).

To return data in an application message buffer:

- set DATA as the address of the buffer (that is, non_NULL, not -1)
- set BUFFLEN to the length of the buffer

If the value of BUFFLEN is less than the length of the message data, behavior depends on whether Accept Truncated Message in the policy receive attributes is selected. If Accept Truncated Message is selected, the data is truncated and there is an AMRC_MSG_TRUNCATED warning. If Accept Truncated Message is not selected (the default), the receive fails and there is an AMRC_RECEIVE_BUFF_LEN_ERR error. To return the data length, set a non_NULL value for DATALEN (that is, not -1).

To return only the data length:

- set DATA to NULL (-1)
- set BUFFLEN to zero
- ensure that Accept Truncated Message in the policy receive attributes is not selected

In this way, you can determine the required buffer size before you issue a second receive request to return the data.

AMRCBRSE (browse selection message)

Browses a message identified by specifying the Correlation ID from the selection message as a selection criterion. See the *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.
01 HPOLICY      PIC S9(9) BINARY.
01 OPTIONS      PIC S9(9) BINARY.
01 HSELMSG      PIC S9(9) BINARY.
01 BUFFLEN      PIC S9(9) BINARY.
01 DATALEN     PIC S9(9) BINARY.
01 DATA        PIC X(n).
01 HRCVMSG      PIC S9(9) BINARY.
01 HRESPONSE    PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

- HRECEIVER** The receiver handle returned by AMSECRRC (input).
- HPOLICY** The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
- OPTIONS** Options controlling the browse operation (input). Possible values are:
 AMBRW-NEXT
 AMBRW-FIRST
 AMBRW-RECEIVE-CURRENT
 AMBRW-DEFAULT (AMBRW-NEXT)
- AMBRW-RECEIVE-CURRENT is equivalent to AMRCRC for the message under the browse cursor.
- HSELMSG** The handle of a selection message object (input). This is used together with the browse options to identify the message to be received (for example, using the Correlation ID). Specify as AMH_NULL_HANDLE to get the next available message. The CCSID, element CCSID, and encoding values from the selection message define the target values for any data conversions. If target conversion values are required without using the Correlation ID for selection, this can be reset (see **AMMSGELC** on page 301) before invoking the **AMRCBRSE** function.
- BUFFLEN** The length in bytes of a buffer in which the data is returned (input).
- DATALEN** The length of the message data, in bytes (output). This can be set to -1 (input).
- DATA** The received message data (output).
- 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).

REASON Reason code (output).

COBOL receiver interface

AMRCCL (close)

Closes the receiver service.

```
CALL 'AMRCCL' USING HRECEIVER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HRECEIVER    PIC S9(9) BINARY.  
01 HPOLICY      PIC S9(9) BINARY.  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HRECEIVER The receiver handle returned by AMSECRRC (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMRCGTD (get definition type)

Gets the definition type of the receiver service.

```
CALL 'AMRCGTD' USING HRECEIVER, TYPE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HRECEIVER    PIC S9(9) BINARY.  
01 TYPE          PIC S9(9) BINARY.  
01 COMPCODE     PIC S9(9) BINARY.  
01 REASON       PIC S9(9) BINARY.
```

HRECEIVER The receiver handle returned by AMSECRRC (input).

TYPE The definition type (output). It can be one of the following:

```
AMDT-UNDEFINED  
AMDT-TEMP-DYNAMIC  
AMDT-DYNAMIC  
AMDT-PREDEFINED
```

Values other than AMDT-UNDEFINED reflect the properties of the underlying queue object.

COMPCODE Completion code (output).

REASON Reason code (output).

AMRCGTLE (get last error)

Gets the information (completion and reason codes) from the last error for the receiver object.

```
CALL 'AMRCGTLE' USING HRECEIVER, BUFFLEN, STRINGLEN, ERRORTXT,
                    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 ERRORTXT    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).

ERRORTXT 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).

REASON Reason code (output).

COBOL receiver interface

AMRCGTQN (get queue name)

Gets the queue name of the receiver service. This is used to determine the queue name of a permanent dynamic receiver service, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. See also AMRCSTQN (set queue name).

```
CALL 'AMRCGTQN' USING HRECEIVER, BUFFLEN, NAMELEN, QUEUENAME,  
                    COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HRECEIVER    PIC S9(9) BINARY.  
01 BUFFLEN     PIC S9(9) BINARY.  
01 NAMELEN     PIC S9(9) BINARY.  
01 QUEUENAME   PIC X(n).  
01 COMPCODE    PIC S9(9) BINARY.  
01 REASON      PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
BUFFLEN	The length in bytes of a buffer in which the queue name is returned (input).
NAMELEN	The length of the queue name, in bytes (output).
QUEUENAME	The queue name of the receiver service (output).
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMRCOP (open)

Opens the receiver service.

```
CALL 'AMRCOP' USING HRECEIVER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HRECEIVER    PIC S9(9) BINARY.  
01 HPOLICY     PIC S9(9) BINARY.  
01 COMPCODE    PIC S9(9) BINARY.  
01 REASON      PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMRCRC (receive)

Receives a message.

```
CALL 'AMRCRC' USING HRECEIVER, HPOLICY, HSELMSG, BUFFLEN, DATALEN, DATA,
                   HRCVMSG, HSENDER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HRECEIVER    PIC S9(9) BINARY.
01 HPOLICY      PIC S9(9) BINARY.
01 HSELMSG      PIC S9(9) BINARY.
01 BUFFLEN      PIC S9(9) BINARY.
01 DATALEN     PIC S9(9) BINARY.
01 DATA        PIC X(n) .
01 HRCVMSG      PIC S9(9) BINARY.
01 HSENDER      PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
HSELMSG	The handle of a selection message object (input). This is used to identify the message to be received (for example, using the correlation ID). Specify as AMH-NULL-HANDLE to get the next available message with no selection.
BUFFLEN	The length in bytes of a buffer in which the data is returned (input).
DATALEN	The length of the message data, in bytes (output). Can be specified as -1 (input).
DATA	The received message data (output).
HRCVMSG	The handle of the message object for the received message (input). If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-RCV-MSG-HANDLE) is used. The message object is reset implicitly before the receive takes place.
HSENDER	The handle of the response sender service that a response message must be sent to, if this is a request message (input). This sender service must have been created without a repository definition, and used exclusively for sending a response. Its definition type must be AMDT-UNDEFINED (it will be set to AMDT-RESPONSE by this call).
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

You can return data in the message object or in an application buffer.

To return the data in the message object (HRCVMSG), rather than the application message buffer, set BUFFLEN to zero and set both DATA and DATALEN as non_NULL (not -1).

To return data in an application message buffer:

- set DATA as the address of the buffer (that is, non_NULL, not -1)
- set BUFFLEN to the length of the buffer

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

correlation ID). Specify as AMH-NULL-HANDLE to get the next available message with no selection. The CCSID, element CCSID, and encoding values from the selection message define the target values for any data conversions. If target conversion values are required without using the Correlation ID for selection, this can be reset (see AMMSSTCI on page 312) before invoking the AMRRCFL function.

DIRNAMELEN	Reserved, must be specified as zero (input). .
DIRNAME	Reserved. .
FILENAMELEN	The length of the file name in bytes (input). .
FILENAME	The name of the file into which the transferred data is to be received (input). This can include a directory prefix to define a fully-qualified or relative file name. If blank then the AMI will use the name of the originating file (including any directory prefix) exactly as it was supplied on the send file call. Note that the original file name may not be appropriate for use by the receiver, either because a path name included in the file name is not applicable to the receiving system, or because the sending and receiving systems use different file naming conventions.
HRCVMSG	The handle of the message object to use to receive the file. This parameter is updated with the message properties, for example the Message ID. If the message is a file message, HRCVMSG receives the message data. If HRCVMSG is specified as AMH-NULL-HANDLE, the default message object (constant AMSD-RCV-MSG-HANDLE) is used. The message object is reset implicitly before the receive takes place.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMRCSTQN (set queue name)

Sets the queue name of the receiver service, when this encapsulates a model queue. This can be used to specify the queue name of a recreated permanent dynamic receiver service, in order to receive messages in a session subsequent to the one in which it was created. See also AMRCGTQN (get queue name).

```
CALL 'AMRCSTQN' USING HRECEIVER, NAMELEN, QUEUENAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HRECEIVER    PIC S9(9) BINARY.
01 NAMELEN     PIC S9(9) BINARY.
01 QUEUENAME   PIC X(n) .
01 COMPCODE    PIC S9(9) BINARY.
01 REASON      PIC S9(9) BINARY.
```

HRECEIVER	The receiver handle returned by AMSECRRC (input).
NAMELEN	The length of the queue name, in bytes (input).
QUEUENAME	The queue name of the receiver service (input).
COMPCODE	Completion code (output).
REASON	Reason code (output).

Distribution list interface functions

A *distribution list* object encapsulates a list of sender objects.

AMDLCLEC (clear error codes)

Clears the error codes in the distribution list object.

```
CALL 'AMDLCLEC' USING HDISTLIST, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HDISTLIST    PIC S9(9) BINARY.  
01 COMPCODE    PIC S9(9) BINARY.  
01 REASON      PIC S9(9) BINARY.
```

HDISTLIST The distribution list handle returned by AMSECRDL (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMDLCCL (close)

Closes the distribution list.

```
CALL 'AMDLCCL' 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).

AMDLCGTL (get last error)

Gets the information (completion and reason codes) from the last error in the distribution list object.

```
CALL 'AMDLCGTL' USING HDISTLIST, BUFFLEN, STRINGLEN, ERRORTXT,  
                     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 ERRORTXT    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).

ERRORTXT 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 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).

REASON Reason code (output).

AMDLSN (send)

Sends a message to each sender in the distribution list.

```
CALL 'AMDLSN' USING HDISTLIST, HPOLICY, HRECEIVER, DATALEN, DATA,
                    HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HDISTLIST    PIC S9(9) BINARY.
01 HPOLICY      PIC S9(9) BINARY.
01 HRECEIVER    PIC S9(9) BINARY.
01 DATALEN     PIC S9(9) BINARY.
01 DATA        PIC X(n).
01 HMSG         PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).
HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
HRECEIVER	The handle of the receiver service to which the response to this message should be sent, if the message being sent is a request message (input). Specify as AMH-NULL-HANDLE if no response is required.
DATALEN	The length of the message data in bytes (input). If specified as zero, any message data will be passed in the message object (HMSG).
DATA	The message data, if DATALEN is non-zero (input).
HMSG	The handle of a message object that specifies the properties of the message being sent (input). If DATALEN is zero, the message object can also contain the message data. If HMSG is specified as AMH-NULL-HANDLE, the default send message object (constant: AMSD-SND-MSG-HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

AMDLSNFL (send file)

Sends data from a file to each sender in the distribution list.

```
CALL 'AMDLSNFL' USING HDISTLIST, HPOLICY, OPTIONS, DIRNAMELEN,
                    DIRNAME, FILENAMELEN, FILENAME, HMSG,
                    COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HDISTLIST    PIC S9(9) BINARY.
01 HPOLICY      PIC S9(9) BINARY.
01 OPTIONS      PIC S9(9) BINARY.
01 DIRNAMELEN   PIC S9(9) BINARY.
01 DIRNAME      PIC X(n).
01 FILENAMELEN  PIC S9(9) BINARY.
01 FILENAME     PIC X(n).
01 HMSG         PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HDISTLIST	The distribution list handle returned by AMSECRDL (input).
------------------	--

COBOL distribution list interface

HPOLICY	The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.
OPTIONS	Reserved, must be specified as zero (input).
DIRNAMELEN	Reserved, must be specified as zero (input).
DIRNAME	Reserved.
FILENAMELEN	The length of the file name in bytes (input).
FILENAME	The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with a receive file call (see "AMRCRCFL (receive file)" on page 330 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
HMSG	The handle of the message object to use to send the file (input). This can be used to specify the Correlation ID for example. If specified as ANM_NULL_HANDLE, the default send message object (constant: AMSD_SND_MSG_HANDLE) is used.
COMPCODE	Completion code (output).
REASON	Reason code (output).

Usage notes

If, in your application, you have previously used a message object, referenced by either handle or name, to send or receive data (including AMI elements or topics), you will need to explicitly call AMMSRS (reset message) before re-using the object for sending a file. This applies even if you use the system default message object handle (constant: AMSD-SND-MSG-HANDLE).

The system default message object handle is used when you set HMSG to AMH-NULL-HANDLE.

Publisher interface functions

A *publisher* object encapsulates a sender object. It provides support for publish messages to a publish/subscribe broker.

AMPBCLEC (clear error codes)

Clears the error codes in the publisher object.

```
CALL 'AMPBCLEC' USING HPUBLISHER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HPUBLISHER    PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HPUBLISHER The publisher handle returned by AMSECRPB (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMPBCL (close)

Closes the publisher service.

```
CALL 'AMPBCL' USING HPUBLISHER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HPUBLISHER    PIC S9(9) BINARY.
01 HPOLICY       PIC S9(9) BINARY.
01 COMPCODE     PIC S9(9) BINARY.
01 REASON       PIC S9(9) BINARY.
```

HPUBLISHER The publisher handle returned by AMSECRPB (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMPBGTC (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 'AMPBGTC' 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).

AMPBGTLT (get last error)

Gets the information (completion and reason codes) from the last error for the publisher object.

```
CALL 'AMPBGTLT' USING HPUBLISHER, BUFFLEN, STRINGLEN, ERRORTXT,  
                     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 ERRORTXT      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).

ERRORTXT 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 AMPBGTLT function call has itself detected an error and failed.

AMPBGTNA (get name)

Gets the name of the publisher service.

```
CALL 'AMPBGTNA' USING HPUBLISHER, BUFFLEN, NAMELEN, NAME,
                    COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HPUBLISHER    PIC S9(9) BINARY.
01 BUFFLEN       PIC S9(9) BINARY.
01 NAMELEN       PIC S9(9) BINARY.
01 NAME          PIC X(n).
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HPUBLISHER The publisher handle returned by AMSECRPB (input).

BUFFLEN The length in bytes of a buffer in which the name is returned (input).

NAMELEN The length of the name, in bytes (output).

NAME The publisher object name (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMPBOP (open)

Opens the publisher service.

```
CALL 'AMPBOP' USING HPUBLISHER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HPUBLISHER    PIC S9(9) BINARY.
01 HPOLICY       PIC S9(9) BINARY.
01 COMPCODE      PIC S9(9) BINARY.
01 REASON        PIC S9(9) BINARY.
```

HPUBLISHER The publisher handle returned by AMSECRPB (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

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).
REASON	Reason code (output).

Subscriber interface functions

A *subscriber* object encapsulates both a sender object and a receiver object. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

AMSBCLEC (clear error codes)

Clears the error codes in the subscriber object.

```
CALL 'AMSBCLEC' USING HSUBSCRIBER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSUBSCRIBER PIC S9(9) BINARY.
01 COMPCODE   PIC S9(9) BINARY.
01 REASON     PIC S9(9) BINARY.
```

HSUBSCRIBER The subscriber handle returned by AMSECRSB (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSBCCL (close)

Closes the subscriber service.

```
CALL 'AMSBCCL' 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).

COBOL subscriber interface

AMSBGTCC (get CCSID)

Gets the coded character set identifier of the subscriber's sender service. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the subscriber must perform CCSID conversion of the message before it is sent.

```
CALL 'AMSBGTCC' USING HSUBSCRIBER, CCSID, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSUBSCRIBER PIC S9(9) BINARY.  
01 CCSID       PIC S9(9) BINARY.  
01 COMPCODE   PIC S9(9) BINARY.  
01 REASON     PIC S9(9) BINARY.
```

HSUBSCRIBER The subscriber handle returned by AMSECRSB (input).

CCSID The coded character set identifier (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSBGTDT (get definition type)

Gets the definition type of the subscriber's receiver service.

```
CALL 'AMSBGTDT' USING HSUBSCRIBER, TYPE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSUBSCRIBER PIC S9(9) BINARY.  
01 TYPE        PIC S9(9) BINARY.  
01 COMPCODE   PIC S9(9) BINARY.  
01 REASON     PIC S9(9) BINARY.
```

HSUBSCRIBER The subscriber handle returned by AMSECRSB (input).

TYPE The definition type (output). It can be:

```
AMDT-UNDEFINED  
AMDT-TEMP-DYNAMIC  
AMDT-DYNAMIC  
AMDT-PREDEFINED
```

COMPCODE Completion code (output).

REASON Reason code (output).

AMSBGTEN (get encoding)

Gets the value used to encode numeric data types for the subscriber's sender service. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the subscriber must convert the encoding of the message before it is sent.

```
CALL 'AMSBGTEN' USING HSUBSCRIBER, ENCODING, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSUBSCRIBER PIC S9(9) BINARY.
01 ENCODING    PIC S9(9) BINARY.
01 COMPCODE   PIC S9(9) BINARY.
01 REASON     PIC S9(9) BINARY.
```

HSUBSCRIBER The subscriber handle returned by AMSECRSB (input).

ENCODING The encoding (output).

COMPCODE Completion code (output).

REASON Reason code (output).

AMSBGTLE (get last error)

Gets the information (completion and reason codes) from the last error for the subscriber object.

```
CALL 'AMSBGTLE' USING HSUBSCRIBER, BUFFLEN, STRINGLEN, ERRORTXT,
                     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 ERRORTXT   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).

ERRORTXT 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).

REASON Reason code (output).

AMSBOP (open)

Opens the subscriber service.

```
CALL 'AMSBOP' USING HSUBSCRIBER, HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSUBSCRIBER PIC S9(9) BINARY.
01 HPOLICY     PIC S9(9) BINARY.
01 COMPCODE   PIC S9(9) BINARY.
01 REASON     PIC S9(9) BINARY.
```

HSUBSCRIBER The subscriber handle returned by AMSECRSB (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

AMSBRC (receive)

Receives a message, normally a publication, using the subscriber service. The message data, topic and other elements can be accessed using the message interface functions (see page 297).

The message data is passed in the message object. There is no option to pass it as a separate parameter as with AMRCRC (this would not give any performance improvement because the MQRFH header has to be removed from the message data after receiving it).

```
CALL 'AMSBRC' USING HSUBSCRIBER, HPOLICY, HSELMSG, HRCVMSG,
                  COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSUBSCRIBER PIC S9(9) BINARY.
01 HPOLICY     PIC S9(9) BINARY.
01 HSELMSG     PIC S9(9) BINARY.
01 HRCVMSG     PIC S9(9) BINARY.
01 COMPCODE   PIC S9(9) BINARY.
01 REASON     PIC S9(9) BINARY.
```

HSUBSCRIBER The subscriber handle returned by AMSECRSB (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

HSELMSG The handle of a selection message object (input). This is used to identify the message to be received (for example, using the correlation ID). Specify as AMH-NULL-HANDLE to get the next available message with no selection.

HRCVMSG The handle of the message object for the received message (input). If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-RCV-MSG-HANDLE) is used. The message object is reset implicitly before the receive takes place.

COMPCODE Completion code (output).

REASON Reason code (output).

COBOL subscriber interface

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).

AMSBBSB (subscribe)

Sends a subscribe message to a publish/subscribe broker using the subscriber service, to register a subscription. The topic and other elements can be specified using the message interface functions (see page 297) before sending the message.

Publications matching the subscription are sent to the receiver service associated with the subscriber. By default, this has the same name as the subscriber service, with the addition of the suffix '.RECEIVER'.

```
CALL 'AMSBBSB' 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 297) before sending the message.

To deregister all topics, a policy providing this option must be specified (this is not the default policy). Otherwise, to remove a previous subscription the topic information specified must match that specified on the relevant AMSBSB request.

```
CALL 'AMSBUN' USING HSUBSCRIBER, HPOLICY, HRECEIVER, HUNSUBMSG,
                  COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HSUBSCRIBER PIC S9(9) BINARY.
01 HPOLICY     PIC S9(9) BINARY.
01 HRECEIVER  PIC S9(9) BINARY.
01 HUNSUBMSG  PIC S9(9) BINARY.
01 COMPCODE   PIC S9(9) BINARY.
01 REASON     PIC S9(9) BINARY.
```

HSUBSCRIBER The subscriber handle returned by AMSECRSB (input).

HPOLICY The handle of a policy (input). If specified as AMH-NULL-HANDLE, the system default policy (constant: AMSD-POL-HANDLE) is used.

HRECEIVER The handle of the receiver service to which the response to this subscribe request should be sent (input). Specify as AMH-NULL-HANDLE if no response is required.

HUNSUBMSG The handle of a message object for the unsubscribe message (input). If specified as AMH-NULL-HANDLE, the default message object (constant: AMSD-SND-MSG-HANDLE) is used.

COMPCODE Completion code (output).

REASON Reason code (output).

Policy interface functions

A *policy* object encapsulates the set of options used for each AMI request (open, close, send, receive, publish and so on). Examples are the priority and persistence of the message, and whether the message is included in a unit of work.

AMPOCLEC (clear error codes)

Clears the error codes in the policy object.

```
CALL 'AMPOCLEC' USING HPOLICY, COMPCODE, REASON.
```

Declare the parameters as follows:

```
01 HPOLICY      PIC S9(9) BINARY.
01 COMPCODE    PIC S9(9) BINARY.
01 REASON      PIC S9(9) BINARY.
```

HPOLICY The policy handle returned by AMSECRPO (input).

COMPCODE Completion code (output).

REASON Reason code (output).

AMPOGTLE (get last error)

Gets the information (completion and reason codes) from the last error for the policy object.

```
CALL 'AMPOGTLE' USING HPOLICY, BUFFLEN, STRINGLEN, ERRORTXT,
                    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 ERRORTXT     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).

ERRORTXT 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).

REASON Reason 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).

REASON Reason code (output).

Part 5. The Java interface

Chapter 13. Using the Application Messaging Interface in Java 355

Structure of the AMI	355
Base classes	355
Interface and helper classes.	356
Exception classes	356
Using the repository	356
System default objects	356
Writing applications in Java	357
Creating and opening objects	357
Sending messages	357
Sample program	359
Receiving messages	359
Sample program	360
Request/response messaging	360
Sample programs	361
File transfer	361
Publish/subscribe messaging	361
Sample programs	362
Using AmElement objects	362
Error handling	363
Transaction support	364
Sending group messages	365
Other considerations	365
Multithreading	365
Using MQSeries with the AMI.	365
Field limits	365
Building Java applications	366
AMI package for Java	366
Running Java programs	366

Chapter 14. Java interface overview 369

Base classes	369
Helper classes	369
Exception classes	369
AmSessionFactory	370
Constructor	370
Session factory management	370
Create session	370
AmSession	371
Session management	371
Create objects	371
Transactional processing.	371
Error handling	371
AmMessage	372
Get values	372
Set values	372
Reset values	372
Read and write data	372
Publish/subscribe filters.	372
Publish/subscribe topics.	373
Publish/subscribe name/value elements	373
Error handling	373
AmSender	374
Open and close.	374
Send	374

Send file	374
Get values	374
Error handling	374
AmReceiver	375
Open and close.	375
Receive and browse	375
Receive file	375
Get values	375
Set value	375
Error handling	375
AmDistributionList	376
Open and close.	376
Send	376
Send file	376
Get values	376
Error handling	376
AmPublisher	377
Open and close.	377
Publish	377
Get values	377
Error handling	377
AmSubscriber	378
Open and close.	378
Broker messages	378
Get values	378
Set value	378
Error handling	378
AmPolicy.	379
Policy management	379
Error handling	379
Helper classes	380
AmConstants	380
AmElement	380
AmObject	380
AmStatus.	380
Exception classes	381
AmException	381
AmErrorException.	381
AmWarningException	381

Chapter 15. Java interface reference 383

Base classes	383
Helper classes	383
Exception classes	383
AmSessionFactory	384
AmSessionFactory.	384
createSession	384
getFactoryName	384
getLocalHost	384
getRepository	384
getTraceLevel	384
getTraceLocation	384
setLocalHost.	384
setRepository	385
setTraceLevel	385
setTraceLocation	385

AmSession	386	enableWarnings	396
begin	386	getCCSID.	396
clearErrorCodes	386	getEncoding	396
close	386	getLastErrorStatus	397
commit	386	getName	397
createDistributionList	386	open	397
createMessage	387	send	397
createPolicy	387	sendFile	398
createPublisher	387	AmReceiver	399
createReceiver	387	browse	399
createSender	387	clearErrorCodes	400
createSubscriber	387	close	400
enableWarnings	388	enableWarnings	400
getLastErrorStatus	388	getDefinitionType	400
getName	388	getLastErrorStatus	401
getTraceLevel	388	getName	401
getTraceLocation	388	getQueueName	401
open	388	open	401
rollback	388	receive	401
AmMessage	389	receiveFile	402
addElement	389	setQueueName	402
addFilter	390	AmDistributionList	403
addTopic	390	clearErrorCodes	403
clearErrorCodes	390	close	403
deleteElement	390	enableWarnings	403
deleteFilter	390	getLastErrorStatus	403
deleteNamedElement	390	getName	403
deleteTopic	391	getSender	403
enableWarnings	391	getSenderCount	403
getCCSID	391	open	403
getCorrelationId	391	send	404
getDataLength	391	sendFile	404
getDataOffset	391	AmPublisher	405
getElement	391	clearErrorCodes	405
getElementCount	391	close	405
getEncoding	392	enableWarnings	405
getFilter	392	getCCSID	405
getFilterCount	392	getEncoding	405
getFormat	392	getLastErrorStatus	405
getGroupStatus	392	getName	405
getLastErrorStatus	392	open	406
getMessageId	393	publish	406
getName	393	AmSubscriber	407
getNamedElement	393	clearErrorCodes	407
getNamedElementCount	393	close	407
getReportCode	393	enableWarnings	407
getTopic	393	getCCSID	407
getTopicCount	393	getDefinitionType	407
getType	394	getEncoding	407
readBytes	394	getLastErrorStatus	407
reset	394	getName	408
setCCSID	394	getQueueName	408
setCorrelationId	394	open	408
setDataOffset	394	receive	408
setEncoding	395	setQueueName	408
setFormat	395	subscribe	409
setGroupStatus	395	unsubscribe	409
writeBytes	395	AmPolicy	410
AmSender	396	clearErrorCodes	410
clearErrorCodes	396	enableWarnings	410
close	396	getLastErrorStatus	410

getName	410
getWaitTime	410
setWaitTime	410
AmConstants	411
AmElement	412
AmElement	412
getName	412
getValue	412
getVersion	412
setVersion	412
toString	412
AmObject	413
clearErrorCodes	413
getLastErrorStatus	413
getName	413
AmStatus	414
AmStatus	414
getCompletionCode	414
getReasonCode	414
getReasonCode2	414
toString	414
AmException	415
getClassName	415
getCompletionCode	415
getMethodName	415
getReasonCode	415
getSource	415
toString	415
AmErrorException	416
getClassName	416
getCompletionCode	416
getMethodName	416
getReasonCode	416
getSource	416
toString	416
AmWarningException	417
getClassName	417
getCompletionCode	417
getMethodName	417
getReasonCode	417
getSource	417
toString	417

Chapter 13. Using the Application Messaging Interface in Java

The Application Messaging Interface for Java (amJava) provides a Java style of programming, while being consistent with the object-style interface of the Application Messaging Interface for C. It uses a Java Native Interface (JNI) library, so it cannot be used to write Applets to run in a browser environment.

This chapter describes the following:

- “Structure of the AMI”
- “Writing applications in Java” on page 357
- “Building Java applications” on page 366

Note that the term *object* is used in this book in the object-oriented programming sense, not in the sense of MQSeries ‘objects’ such as channels and queues.

Structure of the AMI

The following classes are provided:

Base classes

AmSessionFactory	Creates AmSession objects.
AmSession	Creates objects within the AMI session, and controls transactional support.
AmMessage	Contains the message data, message ID and correlation ID, and options that are used when sending or receiving a message (most of which come from the policy definition).
AmSender	This is a service that represents a destination (such as an MQSeries queue) to which messages are sent.
AmReceiver	This is a service that represents a source (such as an MQSeries queue) from which messages are received.
AmDistributionList	Contains a list of sender services to provide a list of destinations.
AmPublisher	Contains a sender service where the destination is a publish/subscribe broker.
AmSubscriber	Contains a sender service (to send subscribe and unsubscribe messages to a publish/subscribe broker) and a receiver service (to receive publications from the broker).
AmPolicy	Defines how the message should be handled, including items such as priority, persistence, and whether it is included in a unit of work.

Structure of the AMI

Interface and helper classes

AmObject	This is a Java interface, which is implemented by the base classes listed previously (with the exception of AmSessionFactory).
AmConstants	This encapsulates all of the constants needed by amJava.
AmElement	This encapsulates name/value pairs that can be added to AmMessage objects.
AmStatus	This encapsulates the error status of amJava objects.

Exception classes

AmException	This is the base Exception class for amJava; all other amJava Exceptions inherit from this class.
AmErrorException	An Exception of this type is raised when an amJava object experiences an error with a severity level of FAILED (CompletionCode = AMCC_FAILED).
AmWarningException	An Exception of this type is raised when an amJava object experiences an error with a severity level of WARNING (CompletionCode = AMCC_WARNING), provided that warnings have been enabled using the enableWarnings method.

Using the repository

You can run AMI applications with or without a repository. If you don't have a repository, you can create an object by specifying its name in a method. It will be created using the appropriate system provided definition (see "System provided definitions" on page 470).

If you have a repository, and you specify the name of an object in a method that matches a name in the repository, the object will be created using the repository definition. (If no matching name is found in the repository, the system provided definition will be used.)

System default objects

The set of system default objects created in C is not accessible directly in Java, but the SYSTEM.DEFAULT.POLICY (constant: AMSD_POL) is used to provide default behavior when a policy is not specified. Objects with identical properties to the system default objects can be created for use in Java using the built-in definitions (see "System provided definitions" on page 470).

Writing applications in Java

This section gives a number of examples showing how to access the Application Messaging Interface using Java.

Many of the method calls are overloaded and in some cases this results in default objects being used. One example of this is the `AmPolicy` object which can be passed on many of the methods. For example:

Method overloading

```
mySender.send(mySendMessage, myPolicy);

mySender.send(mySendMessage);
```

If a policy has been created to provide specific send behavior, use the first example. However, if the default policy is acceptable, use the second example.

The defaulting of behavior using method overloading is used throughout the examples.

Creating and opening objects

Before using the AMI, you must create and open the required objects. Objects are created with names, which might correspond to named objects in the repository. In the case of the creation of a response sender (`myResponder`) in the following example, the default name for a response type object is specified using the `AmConstants` helper class, so the object is created with default responder values.

Creating AMI objects

```
mySessionFactory = new AmSessionFactory("MY.SESSION.FACTORY");
mySession = mySessionFactory.createSession("MY.SESSION");
myPolicy = mySession.createPolicy("MY.POLICY");

mySender = mySession.createSender("AMT.SENDER.QUEUE");
myReceiver = mySession.createReceiver("AMT.RECEIVER.QUEUE");
myResponder = mySession.createSender(AmConstants.AMDEF_RSP_SND);

mySendMessage = mySession.createMessage("MY.SEND.MESSAGE");
myReceiveMessage = mySession.createMessage("MY.RECEIVE.MESSAGE");
```

The objects are then opened. In the following examples, the session object is opened with the default policy, whereas the sender and receiver objects are opened with a specified policy (`myPolicy`).

Opening the AMI objects

```
mySession.open();
mySender.open(myPolicy);
myReceiver.open(myPolicy);
```

Sending messages

The examples in this section show how to send a datagram (send and forget) message. First, the message data is written to the `mySendMessage` object. Data is

Writing applications in Java

always sent in byte form, so the Java `getBytes` method is used to extract the String data as bytes before adding to the message.

Writing data to a message object

```
String dataSent = new String("message to be sent");
mySendMessage.writeBytes(dataSent.getBytes());
```

Next, the message is sent using the sender service `mySender`.

Sending a message

```
mySender.send(mySendMessage);
```

The policy used is either the default policy for the service, if specified, or the system default policy. The message attributes are set from the policy or service, or the default for the messaging transport.

When more control is needed you can pass a policy object:

Sending a message with a specified policy

```
mySender.send(mySendMessage, myPolicy);
```

The policy controls the behavior of the send command. In particular, the policy specifies whether the send is part of a unit of work, the priority, persistence and expiry of the message and whether policy components should be invoked. Whether the queue should be implicitly opened and left open can also be controlled.

To send a message to a distribution list, for instance `myDistList`, use it as the sender service:

Sending a message to a distribution list

```
myDistList.send(mySendMessage);
```

You can set an attribute such as the *Format* before the message is sent, to override the default in the policy or service.

Setting an attribute in a message

```
mySendMessage.setFormat(myFormat);
```

Similarly, after a message has been sent you can retrieve an attribute such as the *MessageID*.

Getting an attribute from a message

```
msgId = mySendMessage.getMessageId();
```

For details of the message attributes that you can set and get, see “AmMessage” on page 372 .

When a message object is used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see **reset** on page 394) and rebuild it each time.

Sample program

For more details, refer to the `SendAndForget.java` sample program (see “Sample programs for AS/400, UNIX, and Windows” on page 461).

Receiving messages

The next example shows how to receive a message from the receiver service `myReceiver`, and to read the data from the message object `myReceiveMessage`.

Receiving a message and retrieving the data

```
myReceiver.receive(myReceiveMessage);  
data = myReceiveMessage.readBytes(myReceiveMessage.getDataLength());
```

The policy used will be the default for the service if defined, or the system default policy. Greater control of the behavior of the receive can be achieved by passing a policy object.

Receiving a message with a specified policy

```
myReceiver.receive(myReceiveMessage, myPolicy);
```

The policy can specify the wait interval, whether the call is part of a unit of work, whether the message should be code page converted, whether all the members of a group must be there before any members can be read, and how to deal with backout failures.

To receive a specific message using its correlation ID, create a selection message object and set its *CorrelId* attribute to the required value. The selection message is then passed as a parameter on the receive.

Receiving a specific message using the correlation ID

```
mySelectionMessage = mySession.createMessage("MY.SELECTION.MESSAGE");  
mySelectionMessage.setCorrelationId(myCorrelId);  
myReceiver.receive(myReceiveMessage, mySelectionMessage, myPolicy);
```

As before, the policy is optional.

You can view the attributes of the message just received, such as the *Encoding*.

Getting an attribute from the message

```
encoding = myReceiveMessage.getEncoding();
```


Writing applications in Java

Sample program

For more details, refer to the `Receiver.java` sample program (see “Sample programs for AS/400, UNIX, and Windows” on page 461).

Request/response messaging

In the *request/response* style of messaging, a requester (or client) application sends a request message and expects to receive a response message back. The responder (or server) application receives the request message and produces the response message (or messages) which it sends back to the requester application. The responder application uses information in the request message to know how to send the response message back to the requester.

In the following examples ‘my’ refers to the requesting application (the client); ‘your’ refers to the responding application (the server).

The requester sends a message as described in “Sending messages” on page 357, specifying the service (`myReceiver`) to which the response message should be sent.

Sending a request message

```
mySender.send(mySendMessage, myReceiver);
```

A policy object can also be specified if required.

The responder receives the message as described in “Receiving messages” on page 359, using its receiver service (`yourReceiver`). It also receives details of the response service (`yourResponder`) for sending the response.

Receiving the request message

```
yourReceiver.receive(yourReceiveMessage, yourResponder);
```

A policy object can be specified if required, as can a selection message object (see “Receiving messages” on page 359).

The responder sends its response message (`yourReplyMessage`) to the response service, specifying the received message to which this is a response.

Sending a response to the request message

```
yourResponder.send(yourReplyMessage, yourReceiveMessage);
```

Finally, the requester application receives the response (`myResponseMessage`), which is correlated with the original message it sent (`mySendMessage`).

Receiving the response message

```
myReceiver.receive(myResponseMessage, mySendMessage);
```

In a typical application, the responder might be a server operating in a loop, receiving requests and replying to them. In this case, the message objects should be set to their initial state and the data cleared before servicing the next request. This is achieved as follows:

Resetting the message object

```
yourReceiveMessage.reset();
yourResponseMessage.reset();
```

Sample programs

For more details, refer to the `Client.java` and `Server.java` sample programs (see “Sample programs for AS/400, UNIX, and Windows” on page 461).

File transfer

You can perform file transfers using the `AmSender.sendFile` and `AmReceiver.receiveFile` methods.

Sending a file using the `sendFile` method

```
mySender.sendFile(mySendMessage, myfilename, myPolicy)
```

Receiving a file using the `receiveFile` method

```
myReceiver.receiveFile(myReceiveMessage, myfileName, myPolicy)
```

For a complete description of file transfer, refer to “File transfer” on page 21.

Publish/subscribe messaging

With *publish/subscribe* messaging a *publisher* application publishes messages to *subscriber* applications using a *broker*. The message published contains application data and one or more *topic* strings that describe the data. A subscribing application subscribes to topics informing the broker which topics it is interested in. When the broker receives a message from a publisher it compares the topics in the messages to the topics in the subscription from subscribing applications. If they match, the broker forwards the message to the subscribing application.

Data on a particular topic is published as shown in the next example.

Publishing a message on a specified topic

```
String publicationTopic = new String("Weather");
String publicationData = new String("The weather is sunny");

myPubMessage.addTopic(publicationTopic);
myPubMessage.writeBytes(publicationData.getBytes());
myPublisher.publish(myPubMessage, myReceiver);
```

`myReceiver` identifies a response service to which the broker will send any response messages. You can also specify a policy object to modify the behavior of the command.

Writing applications in Java

To subscribe to a publish/subscribe broker you need to specify one or more topics.

Subscribing to a broker on specified topics

```
String weather = new String("Weather");
String birds = new String("Birds");

mySubMessage.addTopic(weather);
mySubMessage.addTopic(birds);
mySubscriber.subscribe(mySubMessage, myReceiver);
```

Broker response messages will be sent to myReceiver.

To remove a subscription, add the topic or topics to be deleted to the message object, and use:

Removing a subscription

```
mySubscriber.unsubscribe(myUnsubMessage, myReceiver);
```

To receive a publication from a broker, use:

Receiving a publication

```
mySubscriber.receive(myReceiveMessage, myPolicy);
publication = myReceiveMessage.readBytes(
    myReceiveMessage.getDataLength());
```

You can then use the **getTopicCount** and **getTopic** methods to extract the topic or topics from the message object.

Subscribing applications can also exploit content-based publish/subscribe by passing a filter on subscribe and unsubscribe calls (see "Using MQSeries Integrator Version 2" on page 458).

Sample programs

For more details, refer to the `Publisher.java` and `Subscriber.java` sample programs (see "Sample programs for AS/400, UNIX, and Windows" on page 461).

Using AmElement objects

Publish/subscribe brokers (such as MQSeries Publish/Subscribe) respond to messages that contain name/value pairs to define the commands and options to be carried out. The Application Messaging Interface contains some methods which produce these name/value pairs directly (such as **AmSubscriber.subscribe**). For less commonly used commands, the name/value pairs can be added to a message using an `AmElement` object.

For example, to send a message containing a 'Request Update' command, use the following:

Using an AmElement object to construct a command message

```
AmElement bespokeElement = new AmElement("MQPSCCommand", "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.

Writing applications in Java

For example:

```
try
{
    ...
    mySession.enableWarnings(true);
    mySession.open();
    ...
}
catch (AmErrorException errorEx)
{
    AmStatus sessionStatus = mySession.getLastErrorStatus();
    switch (sessionStatus.getReasonCode())
    {
        case AmConstants.AMRC_XXXX:
            ...
        case AmConstants.AMRC_XXXX:
            ...
    }
    mySession.clearErrorCodes();
}
catch (AmWarningException warningEx)
{
    ...
}
```

Because most of the objects implement the `AmObject` interface, a generic error handling routine can be written. For example:

```
try
{
    ...
    mySession.open();
    ...
    mySender.send(myMessage);
    ...
    mySender.send(myMessage);
    ...
    mySession.commit();
}
catch(AmException amex);
{
    AmStatus status;
    status = amex.getSource().getLastErrorStatus();
    System.out.println("Object in error; name="+ amex.getSource().getName());
    System.out.println("Object in error; RC="+ status.getReasonCode());
    ...
    amex.getSource().clearErrorCodes();
}
```

The catch block works because all objects that throw the `AmException` in the try block are `AmObjects`, and so they all have `getName`, `getLastErrorStatus` and `clearErrorCodes` methods.

Transaction support

Messages sent and received by the AMI can, optionally, be part of a transactional unit of work. A message is included in a unit of work based on the setting of the syncpoint attribute specified in the policy used on the call. The scope of the unit of work is the session handle and only one unit of work may be active at any time.

The API calls used to control the transaction depend on the type of transaction that is used.

- MQSeries messages are the only resource used

A transaction is started by the first message sent or received under syncpoint control, as specified in the policy specified for the send or receive. Multiple messages can be included in the same unit of work. The transaction is committed or backed out using the **commit** or **rollback** method.

- MQSeries is used as an XA transaction coordinator
The transaction must be started explicitly using the **begin** method before the first recoverable resource (such as a relational database) is changed. The transaction is committed or backed out using an **commit** or **rollback** method.
- An external transaction coordinator is used
The transaction is controlled using the API calls of an external transaction coordinator (such as CICS, Encina or Tuxedo). The AMI calls are not used but the syncpoint attributed must still be specified in the policy used on the call.

Sending group messages

The AMI allows a sequence of related messages to be included in, and sent as, a message group. Group context information is sent with each message to allow the message sequence to be preserved and made available to a receiving application. To include messages in a group, the group status information of the first and subsequent messages in the group must be set as follows:

```
AMGRP_FIRST_MSG_IN_GROUP for the first message
AMGRP_MIDDLE_MSG_IN_GROUP for all messages other than first and last
AMGRP_LAST_MSG_IN_GROUP for the last message
```

The message status is set using the **AmMessage.setGroupStatus** method. For a complete description of group messages, refer to “Sending group messages” on page 26.

Other considerations

Multithreading

If you are using multithreading with the AMI, a session normally remains locked for the duration of a single AMI call. If you use receive with wait, the session remains locked for the duration of the wait, which might be unlimited (that is, until the wait time is exceeded or a message arrives on the queue). If you want another thread to run while a thread is waiting for a message, it must use a separate session.

AMI handles and object references can be used on a different thread from that on which they were first created for operations that do not involve an access to the underlying (MQSeries) message transport. Functions such as initialize, terminate, open, close, send, receive, publish, subscribe, unsubscribe, and receive publication will access the underlying transport restricting these to the thread on which the session was first opened (for example, using **AmSession.open**). An attempt to issue these on a different thread will cause an error to be returned by MQSeries and a transport error (AMRC_TRANSPORT_ERR) will be reported to the application.

Using MQSeries with the AMI

You must not mix MQSeries function calls with AMI calls within the same process.

Field limits

When string and binary properties such as queue name, message format, and correlation ID are set, the maximum length values are determined by MQSeries, the underlying message transport. See the rules for naming MQSeries objects in the *MQSeries Application Programming Guide*.

Building Java applications

This section contains information that will help you write, prepare, and run your Java application programs on the various operating systems supported by the AMI.

AMI package for Java

AMI provides a jar file that contains all the classes comprising the AMI package for Java.

com.ibm.mq.amt	Java package
com.ibm.mq.amt.jar	Java jar file

This jar file is installed under:

/QIBM/ProdData/mqm/amt/Java/lib	(AS/400)
/java/lib	(UNIX)
\java\lib	(Windows)

See “Directory structure” on page 430 (AIX), page 433 (AS/400), page 438 (HP-UX), page 445 (Solaris), or page 448 (Windows).

To use this package you must:

- Import the package into your Java application by using the following statement in that application:

```
import com.ibm.mq.amt.*;
```
- Make sure that the AMI jar file is in your CLASSPATH environment variable. See “Setting the runtime environment” on page 429 (AIX), on page 432 (AS/400), page 437 (HP-UX), page 444 (Solaris), or page 447 (Windows). Do this both in the environment in which your Java program is compiled, and in the environment in which it is run.

Running Java programs

This section explains what you have to do to prepare and run your Java programs on the AIX, AS/400, HP-UX, Sun Solaris, Windows 98 and Windows NT operating systems.

The AMI interface for Java makes use of JNI (Java Native Interface) and so requires a platform native library to run successfully. This library must be accessible to your runtime environment. See “Language compilers” on page 426 for versions of the Java Developer’s Kit (JDK) supported by the AMI.

AIX Make sure that the JNI library `libamtJava.so` is accessible to your runtime environment. To do this, you should perform:

```
export LIBPATH=$LIBPATH:/usr/mqm/lib:
```

AS/400 Make sure that the library `QMQMAMI` is in the library list.

If you use the AS/400 QShell interpreter, you must use the `export -s` command, so that AMI can access the required environment variables.

HP-UX Make sure that the JNI library `libamtJava.sl` is accessible to your runtime environment. To do this, you should perform:

```
export SHLIB_PATH=$SHLIB_PATH:/opt/mqm/lib:
```

Building Java applications

Solaris Make sure that the JNI library `libamtJava.so` is accessible to your runtime environment. To do this, you should perform:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/mqm/lib:
```

Windows Make sure that the JNI library `amtJava.dll` is in one of the directories specified in the `PATH` environment variable for your runtime environment. For example:

```
SET PATH=%PATH%;C:\MQSeries\bin;
```

If you already have MQSeries installed, it is likely that this environment has already been set up for you.

Once the AMI jar file and the JNI library are referenced in your runtime environment you can run your Java application. For example, to run an application called `mine` that exists in a package `com.xxx.com`, perform:

```
java com.xxx.com.mine
```


Building Java applications

Chapter 14. Java interface overview

This chapter contains an overview of the structure of the Application Messaging Interface for Java. Use it to find out what functions are available in this interface.

The Java interface provides sets of methods for each of the classes listed below. The methods available for each class are listed in the following pages. Follow the page references to see the reference information for each method.

Base classes

AmSessionFactory	page 370
AmSession	page 371
AmMessage	page 372
AmSender	page 374
AmReceiver	page 375
AmDistributionList	page 376
AmPublisher	page 377
AmSubscriber	page 378
AmPolicy	page 379

Helper classes

AmConstants	page 380
AmElement	page 380
AmObject	page 380
AmStatus	page 380

Exception classes

AmException	page 381
AmErrorException	page 381
AmWarningExcpetion	page 381

AmSessionFactory

The `AmSessionFactory` class is used to create `AmSession` objects.

Constructor

Constructor for `AmSessionFactory`.

`AmSessionFactory` page 384

Session factory management

Methods to return the name of an `AmSessionFactory` object, and to control traces.

`getFactoryName` page 384

`getLocalHost` page 384

`getRepository` page 384

`getTraceLevel` page 384

`getTraceLocation` page 384

`setLocalHost` page 384

`setRepository` page 385

`setTraceLevel` page 385

`setTraceLocation` page 385

Create session

Method to create an `AmSession` object.

`createSession` page 384

AmSession

The **AmSession** object creates and manages all other objects, and provides scope for a unit of work.

Session management

Methods to open and close an AmSession object, to return its name, and to control traces.

open	page 388
close	page 386
getName	page 388
getTraceLevel	page 388
getTraceLocation	page 388

Create objects

Methods to create AmMessage, AmSender, AmReceiver, AmDistributionList, AmPublisher, AmSubscriber, and AmPolicy objects.

createMessage	page 387
createSender	page 387
createReceiver	page 387
createDistributionList	page 386
createPublisher	page 387
createSubscriber	page 387
createPolicy	page 387

Transactional processing

Methods to begin, commit and rollback a unit of work.

begin	page 386
commit	page 386
rollback	page 388

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 386
enableWarnings	page 388
getLastErrorStatus	page 388

AmMessage

An **AmMessage** object encapsulates an MQSeries message descriptor (MQMD) structure, and it contains the message data if this is not passed as a separate parameter.

Get values

Methods to get the coded character set ID, correlation ID, encoding, format, group status, message ID and name of the message object.

getCCSID	page 391
getCorrelationId	page 391
getEncoding	page 392
getFormat	page 392
getGroupStatus	page 392
getMessageId	page 393
getName	page 393
getReportCode	page 393
getType	page 394

Set values

Methods to set the coded character set ID, correlation ID, format and group status of the message object.

setCCSID	page 394
setCorrelationId	page 394
setEncoding	page 395
setFormat	page 395
setGroupStatus	page 395

Reset values

Method to reset the message object to the state it had when first created.

reset	page 394
--------------	----------

Read and write data

Methods to read or write byte data to or from the message object, to get and set the data offset, and to get the length of the data.

getDataLength	page 391
getDataOffset	page 391
setDataOffset	page 394
readBytes	page 394
writeBytes	page 395

Publish/subscribe filters

Methods to manipulate filters for content-based publish/subscribe.

addFilter	page 390
deleteFilter	page 390
getFilter	page 392
getFilterCount	page 392

Publish/subscribe topics

Methods to manipulate the topics in a publish/subscribe message.

addTopic	page 390
deleteTopic	page 391
getTopic	page 393
getTopicCount	page 393

Publish/subscribe name/value elements

Methods to manipulate the name/value elements in a publish/subscribe message.

addElement	page 389
deleteElement	page 390
getElement	page 391
getElementCount	page 391
deleteNamedElement	page 390
getNamedElement	page 393
getNamedElementCount	page 393

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 390
enableWarnings	page 391
getLastErrorStatus	page 392

AmSender

An **AmSender** object encapsulates an MQSeries object descriptor (MQOD) structure.

Open and close

Methods to open and close the sender service.

open	page 397
close	page 396

Send

Method to send a message.

send	page 397
-------------	----------

Send file

Method to send data from a file

sendFile	page 398
-----------------	----------

Get values

Methods to get the coded character set ID, encoding and name of the sender service.

getCCSID	page 396
getEncoding	page 396
getName	page 397

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 396
enableWarnings	page 396
getLastErrorStatus	page 397

AmReceiver

An **AmReceiver** object encapsulates an MQSeries object descriptor (MQOD) structure.

Open and close

Methods to open and close the receiver service.

open	page 401
close	page 400

Receive and browse

Methods to receive or browse a message.

receive	page 401
browse	page 399

Receive file

Method to receive file message data into a file.

receiveFile	page 402
--------------------	----------

Get values

Methods to get the definition type, name and queue name of the receiver service.

getDefinitionType	page 400
getName	page 401
getQueueName	page 401

Set value

Method to set the queue name of the receiver service.

setQueueName	page 402
---------------------	----------

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 400
enableWarnings	page 400
getLastErrorStatus	page 401

AmDistributionList

An **AmDistributionList** object encapsulates a list of **AmSender** objects.

Open and close

Methods to open and close the distribution list service.

open page 403

close page 403

Send

Method to send a message to the distribution list.

send page 404

Send file

Method to send data from a file to each sender defined in the distribution list.

sendFile page 404

Get values

Methods to get the name of the distribution list service, a count of the **AmSenders** in the list, and one of the **AmSenders** that is contained in the list.

getName page 403

getSenderCount page 403

getSender page 403

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes page 403

enableWarnings page 403

getLastErrorStatus page 403

AmPublisher

An **AmPublisher** object encapsulates a sender service and provides support for publishing messages to a publish/subscribe broker.

Open and close

Methods to open and close the publisher service.

open	page 406
close	page 405

Publish

Method to publish a message.

publish	page 406
----------------	----------

Get values

Methods to get the coded character set ID, encoding and name of the publisher service.

getCCSID	page 405
getEncoding	page 405
getName	page 405

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 405
enableWarnings	page 405
getLastErrorStatus	page 405

AmSubscriber

An **AmSubscriber** object encapsulates both a sender service and a receiver service. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

Open and close

Methods to open and close the subscriber service.

open	page 408
close	page 407

Broker messages

Methods to subscribe to a broker, remove a subscription, and receive a publication from the broker.

subscribe	page 409
unsubscribe	page 409
receive	page 408

Get values

Methods to get the coded character set ID, definition type, encoding, name and queue name of the subscriber service.

getCCSID	page 407
getDefinitionType	page 407
getEncoding	page 407
getName	page 408
getQueueName	page 408

Set value

Method to set the queue name of the subscriber service.

setQueueName	page 408
---------------------	----------

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 407
enableWarnings	page 407
getLastErrorStatus	page 407

AmPolicy

An **AmPolicy** object encapsulates the options used during AMI operations.

Policy management

Methods to return the name of the policy, and to get and set the wait time when receiving a message.

getName	page 410
getWaitTime	page 410
setWaitTime	page 410

Error handling

Methods to clear the error codes, enable warnings, and return the status from the last error.

clearErrorCodes	page 410
enableWarnings	page 410
getLastErrorStatus	page 410

Helper classes

A Java Interface, and classes that encapsulate constants, name/value elements, and error status.

AmConstants

Provides access to all the AMI constants.

AmConstants page 411

AmElement

Constructor for AmElement, and methods to return the name, type, value and version of an element, to set the version, and to return a String representation of the element.

AmElement page 412

getName page 412

getValue page 412

getVersion page 412

setVersion page 412

toString page 412

AmObject

A Java Interface containing methods to return the name of the object, to clear the error codes and to return the last error condition.

clearErrorCodes page 413

getLastErrorStatus page 413

getName page 413

AmStatus

Constructor for AmStatus, and methods to return the completion code, reason code, secondary reason code and status text, and to return a String representation of the AmStatus.

AmStatus page 414

getCompletionCode page 414

getReasonCode page 414

getReasonCode2 page 414

toString page 414

Exception classes

Classes that encapsulate error and warning conditions. `AmErrorException` and `AmWarningException` inherit from `AmException`.

AmException

Methods to return the completion code and reason code from the `Exception`, the class name, method name and source of the `Exception`, and to return a `String` representation of the `Exception`.

<code>getClassName</code>	page 415
<code>getCompletionCode</code>	page 415
<code>getMethodName</code>	page 415
<code>getReasonCode</code>	page 415
<code>getSource</code>	page 415
<code>toString</code>	page 415

AmErrorException

Methods to return the completion code and reason code from the `Exception`, the class name, method name and source of the `Exception`, and to return a `String` representation of the `Exception`.

<code>getClassName</code>	page 416
<code>getCompletionCode</code>	page 416
<code>getMethodName</code>	page 416
<code>getReasonCode</code>	page 416
<code>getSource</code>	page 416
<code>toString</code>	page 416

AmWarningException

Methods to return the completion code and reason code from the `Exception`, the class name, method name and source of the `Exception`, and to return a `String` representation of the `Exception`.

<code>getClassName</code>	page 417
<code>getCompletionCode</code>	page 417
<code>getMethodName</code>	page 417
<code>getReasonCode</code>	page 417
<code>getSource</code>	page 417
<code>toString</code>	page 417

Java interface overview

Chapter 15. Java interface reference

In the following sections the Java interface methods are listed by the class they refer to. Within each section the methods are listed in alphabetical order.

Note that where constants are shown (for example, `AMRC_NONE`), they can be accessed using the `AmConstants` class (for example, `AmConstants.AMRC_NONE`). See page 411.

Base classes

Note that all of the methods in these classes can throw `AmWarningException` and `AmErrorException` (see below). However, by default, `AmWarningExceptions` are not raised.

AmSessionFactory	page 384
AmSession	page 386
AmMessage	page 389
AmSender	page 396
AmReceiver	page 399
AmDistributionList	page 403
AmPublisher	page 405
AmSubscriber	page 407
AmPolicy	page 410

Helper classes

AmConstants	page 411
AmElement	page 412
AmObject	page 413
AmStatus	page 414

Exception classes

AmException	page 415
AmErrorException	page 416
AmWarningException	page 417

AmSessionFactory

The `AmSessionFactory` class is used to create `AmSession` objects.

AmSessionFactory

Constructor for an `AmSessionFactory`.

```
AmSessionFactory(String name);
```

name The name of the `AmSessionFactory`. This is the location of the data files used by the AMI (the repository file and the local host file). The name can be a fully qualified directory that includes the path under which the files are located. Otherwise, see “Local host and repository files (AS/400, UNIX, and Windows)” on page 450 for the location of these files.

createSession

Creates an `AmSession` object.

```
AmSession createSession(String name);
```

name The name of the `AmSession`.

getFactoryName

Returns the name of the `AmSessionFactory`.

```
String getFactoryName();
```

getLocalHost

Returns the name of the local host file.

```
String getLocalHost();
```

getRepository

Returns the name of the repository file.

```
String getRepository();
```

getTraceLevel

Returns the trace level for the `AmSessionFactory`.

```
int getTraceLevel();
```

getTraceLocation

Returns the location of the trace for the `AmSessionFactory`.

```
String getTraceLocation();
```

setLocalHost

Sets the name of the AMI local host file to be used by any `AmSession` created from this `AmSessionFactory`. (Otherwise, the default host file `amthost.xml` is used.)

```
void setLocalHost(String fileName);
```

fileName The name of the file used by the AMI as the local host file. This file must be present on the local file system or an error will be produced upon the creation of an `AmSession`.

setRepository

Sets the name of the AMI repository to be used by any AmSession created from this AmSessionFactory. (Otherwise, the default repository file amt.xml is used.)

```
void setRepository(String fileName);
```

fileName

The name of the file used by the AMI as the repository. This file must be present on the local file system or an error will be produced upon the creation of an AmSession.

setTraceLevel

Sets the trace level for the AmSessionFactory.

```
void setTraceLevel(int level);
```

level

The trace level to be set in the AmSessionFactory. Trace levels are 0 through 9, where 0 represents minimal tracing and 9 represents a fully detailed trace.

setTraceLocation

Sets the location of the trace for the AmSessionFactory.

```
void setTraceLocation(String location);
```

location

The location on the local system where trace files will be written. This location must be a directory, and it must exist before the trace is run.

AmSession

An **AmSession** object provides the scope for a unit of work and creates and manages all other objects, including at least one connection object. Each (MQSeries) connection object encapsulates a single MQSeries queue manager connection. The session object definition specifying the required queue manager connection can be provided by a repository policy definition, or by default will name a single local queue manager with no repository. The session, when deleted, is responsible for releasing memory by closing and deleting all other objects that it manages.

begin

Begins a unit of work in this AmSession, allowing an AMI application to take advantage of the resource coordination provided in MQSeries. The unit of work can subsequently be committed by the **commit** method, or backed out by the **rollback** method. This should be used only when AMI is the transaction coordinator. If available, native coordination APIs (for example CICS or Tuxedo) should be used.

begin is overloaded. The `policy` parameter is optional.

```
void begin(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

clearErrorCodes

Clears the error codes in the AmSession.

```
void clearErrorCodes();
```

close

Closes the AmSession, and all open objects owned by it. **close** is overloaded: the `policy` parameter is optional.

```
void close(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

commit

Commits a unit of work that was started by **AmSession.begin**. **commit** is overloaded: the `policy` parameter is optional.

```
void commit(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

createDistributionList

Creates an AmDistributionList object.

```
AmDistributionList createDistributionList(String name);
```

name The name of the AmDistributionList. This must match the name of a distribution list defined in the repository.

createMessage

Creates an AmMessage object.

```
AmMessage createMessage(String name);
```

name The name of the AmMessage. This can be any name that is meaningful to the application.

createPolicy

Creates an AmPolicy object.

```
AmPolicy createPolicy(String name);
```

name The name of the AmPolicy. If it matches a policy defined in the repository, the policy will be created using the repository definition, otherwise it will be created with default values.

createPublisher

Creates an AmPublisher object.

```
AmPublisher createPublisher(String name);
```

name The name of the AmPublisher. If it matches a publisher defined in the repository, the publisher will be created using the repository definition, otherwise it will be created with default values (that is, with an AmSender name that matches the publisher name).

createReceiver

Creates an AmReceiver object.

```
AmReceiver createReceiver(String name);
```

name The name of the AmReceiver. If it matches a receiver defined in the repository, the receiver will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the receiver name).

createSender

Creates an AmSender object.

```
AmSender createSender(String name);
```

name The name of the AmSender. If it matches a sender defined in the repository, the sender will be created using the repository definition, otherwise it will be created with default values (that is, with a queue name that matches the sender name).

createSubscriber

Creates an AmSubscriber object.

```
AmSubscriber createSubscriber(String name);
```

name The name of the AmSubscriber. If it matches a subscriber defined in the repository, the subscriber will be created using the repository definition, otherwise it will be created with default values (that is, with an AmSender name that matches the subscriber name, and an AmReceiver name that is the same with the addition of the suffix '.RECEIVER').

Java AmSession

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void enableWarnings(boolean warningsOn);
```

warningsOn If set to true, AmWarningExceptions will be raised for this object.

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmSession.

```
String getName();
```

getTraceLevel

Returns the trace level of the AmSession.

```
int getTraceLevel();
```

getTraceLocation

Returns the location of the trace for the AmSession.

```
String getTraceLocation();
```

open

Opens an AmSession using the specified policy. The application profile group of this policy provides the connection definitions enabling the connection objects to be created. The specified library is loaded for each connection and its dispatch table initialized. If the transport type is MQSeries and the MQSeries local queue manager library cannot be loaded, the MQSeries client queue manager is loaded. Each connection object is then opened.

open is overloaded: the policy parameter is optional.

```
void open(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

rollback

Rolls back a unit of work that was started by **AmSession.begin**, or under policy control. **rollback** is overloaded: the policy parameter is optional.

```
void rollback(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmMessage

An **AmMessage** object encapsulates the MQSeries MQMD message properties, and name/value elements such as the topics for publish/subscribe messages. In addition it contains the application data.

The initial state of the message object is:

CCSID	default queue manager CCSID
correlationId	all zeros
dataLength	zero
dataOffset	zero
elementCount	zero
encoding	AMENC_NATIVE
format	AMFMT_STRING
groupStatus	AMGRP_MSG_NOT_IN_GROUP
reportCode	AMFB_NONE
topicCount	zero
type	AMMT_DATAGRAM

When a message object is used to send a message, it might not be left in the same state as it was before the send. Therefore, if you use the message object for repeated send operations, it is advisable to reset it to its initial state (see **reset** on page 394) and rebuild it each time.

Note that the following methods are only valid after a session has been opened with **AmSession.open**:

addElement	page 389
deleteElement	page 390
getElement	page 391
getElementCount	page 391
deleteNamedElement	page 390
getNamedElement	page 393
getNamedElementCount	page 393
addTopic	page 390
deleteTopic	page 391
getTopic	page 393
getTopicCount	page 393

addElement

Adds a name/value element to an AmMessage object. **addElement** is overloaded: the **element** parameter is required, but the **options** parameter is optional.

```
void addElement(
    AmElement element,
    int options);
```

element The element to be added to the AmMessage.

options The options to be used. This parameter is reserved and must be set to zero.

Java AmMessage

addFilter

Adds a publish/subscribe filter to an AmMessage object.

```
void addFilter(String filter);
```

filter The filter to be added to the AmMessage.

addTopic

Adds a publish/subscribe topic to an AmMessage object.

```
void addTopic(String topicName);
```

topicName The name of the topic to be added to the AmMessage.

clearErrorCodes

Clears the error in the AmMessage object.

```
void clearErrorCodes();
```

deleteElement

Deletes the element in the AmMessage object at the specified index. Indexing is within all elements of a message, and might include topics (which are specialized elements).

```
void deleteElement(int index);
```

index The index of the element to be deleted, starting from zero. On completion, elements with higher index values than that specified will have those values reduced by one.

getElementCount gets the number of elements in the message.

deleteFilter

Deletes a publish/subscribe filter in an AmMessage object at the specified index. Indexing is within all filters in the message.

```
void deleteFilter(int filterIndex);
```

filterIndex The index of the filter to be deleted, starting from zero.
getFilterCount gets the number of filters in a message.

deleteNamedElement

Deletes the element with the specified name in the AmMessage object, at the specified index. Indexing is within all elements that share the same name.

```
void deleteNamedElement(  
    String name,  
    int index);
```

name The name of the element to be deleted.

index The index of the element to be deleted, starting from zero. On completion, elements with higher index values than that specified will have those values reduced by one.

getNamedElementCount gets the number of elements in the message with the specified name.

deleteTopic

Deletes a publish/subscribe topic in an AmMessage object at the specified index. Indexing is within all topics in the message.

```
void deleteTopic(int index);
```

index The index of the topic to be deleted, starting from zero. **getTopicCount** gets the number of topics in the message.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void enableWarnings(boolean warningsOn);
```

warningsOn If set to true, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier used by AmMessage.

```
int getCCSID();
```

getCorrelationId

Returns the correlation identifier for the AmMessage.

```
byte[] getCorrelationId();
```

getDataLength

Returns the length of the message data in the AmMessage.

```
int getDataLength();
```

getDataOffset

Returns the current offset in the message data for reading or writing data bytes.

```
int getDataOffset();
```

getElement

Returns an element in an AmMessage object at the specified index. Indexing is within all elements in the message, and might include topics (which are specialized elements).

```
AmElement getElement(int index);
```

index The index of the element to be returned, starting from zero. **getElementCount** gets the number of elements in the message.

getElementCount

Returns the total number of elements in an AmMessage object. This might include topics (which are specialized elements).

```
int getElementCount();
```


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();
```

The following values can be returned:

```
AMFB_NONE
AMFB_EXPIRATION
AMFB_COA
AMFB_COD
AMFB_ERROR
```

getTopic

Returns the publish/subscribe topic in the AmMessage object, at the specified index. Indexing is within all topics.

```
String getTopic(int index);
```

index The index of the topic to be returned, starting from zero.
getTopicCount gets the number of topics in the message.

getTopicCount

Returns the total number of publish/subscribe topics in the AmMessage object.

```
int getTopicCount();
```

Java AmMessage

getType

Returns the message type from the AmMessage.

```
int getType();
```

The following values can be returned:

```
AMMT_REQUEST  
AMMT_REPLY  
AMMT_REPORT  
AMMT_DATAGRAM
```

readBytes

Populates a byte array with data from the AmMessage, starting at the current data offset (which must be positioned before the end of the data for the read to be successful). Use **setDataOffset** to specify the data offset. **readBytes** will advance the data offset by the number of bytes read, leaving the offset immediately after the last byte read.

```
byte[] readBytes(int dataLength);
```

dataLength The maximum number of bytes to be read from the message data. The number of bytes returned is the minimum of dataLength and the number of bytes between the data offset and the end of the data.

reset

Resets the AmMessage object to its initial state (see page 389).

reset is overloaded: the options parameter is optional.

```
void reset(int options);
```

options A reserved field that must be set to zero.

setCCSID

Sets the coded character set identifier used by the AmMessage object.

```
void setCCSID(int codedCharSetId);
```

codedCharSetId

The CCSID to be set in the AmMessage.

setCorrelationId

Sets the correlation identifier in the AmMessage object.

```
void setCorrelationId(byte[] correlId);
```

correlId The correlation identifier to be set in the AmMessage. The correlation identifier can be reset by specifying this as a zero length byte array. For example:

```
byte[] myByteArray = new byte[0];  
myMessage.setCorrelationId(myByteArray);
```

setDataOffset

Sets the data offset for reading or writing byte data.

```
void setDataOffset(int dataOffset);
```

dataOffset The data offset to be set in the AmMessage. Set an offset of zero to read or write from the start of the data.

setEncoding

Sets the encoding of the data in the AmMessage object.

```
void setEncoding(int encoding);
```

encoding The encoding to be used in the AmMessage. It can take one of the following values:

```
AMENC_NORMAL
AMENC_NORMAL_FLOAT_390
AMENC_REVERSED
AMENC_REVERSED_FLOAT_390
AMENC_UNDEFINED
```

setFormat

Sets the format for the AmMessage object.

```
void setFormat(String format);
```

format The format to be used in the AmMessage. It can take one of the following values:

```
AMFMT_NONE
AMFMT_STRING
AMFMT_RF_HEADER
```

If set to AMFMT_NONE, the default format for the sender will be used (if available).

setGroupStatus

Sets the group status value for the AmMessage. This indicates whether the message is in a group, and if it is the first, middle, last or only one in the group. Once you start sending messages in a group, you must complete the group before sending any messages that are not in the group.

If you specify AMGRP_MIDDLE_MSG_IN_GROUP or AMGRP_LAST_MSG_IN_GROUP without specifying AMGRP_FIRST_MSG_IN_GROUP, the behavior is the same as for AMGRP_FIRST_MSG_IN_GROUP and AMGRP_ONLY_MSG_IN_GROUP.

If you specify AMGRP_FIRST_MSG_IN_GROUP out of sequence, the behavior is the same as for AMGRP_MIDDLE_MSG_IN_GROUP.

```
void setGroupStatus(int groupStatus);
```

groupStatus The group status to be set in the AmMessage. It can take one of the following values:

```
AMGRP_MSG_NOT_IN_GROUP
AMGRP_FIRST_MSG_IN_GROUP
AMGRP_MIDDLE_MSG_IN_GROUP
AMGRP_LAST_MSG_IN_GROUP
AMGRP_ONLY_MSG_IN_GROUP
```

writeBytes

Writes a byte array into the AmMessage object, starting at the current data offset. If the data offset is not at the end of the data, existing data is overwritten. Use **setDataOffset** to specify the data offset. **writeBytes** will advance the data offset by the number of bytes written, leaving it immediately after the last byte written.

```
void writeBytes(byte[] data);
```

data The data to be written to the AmMessage.

AmSender

An **AmSender** object encapsulates an MQSeries object descriptor (MQOD) structure. This represents an MQSeries queue on a local or remote queue manager. An open sender service is always associated with an open connection object (such as a queue manager connection). Support is also included for dynamic sender services (those that encapsulate model queues). The required sender service object definitions can be provided from a repository, or created without a repository definition by defaulting to the existing queue objects on the local queue manager.

The AmSender object must be created before it can be opened. This is done using **AmSession.createSender**.

A *responder* is a special type of AmSender used for sending a response to a request message. It is not created from a repository definition. Once created, it must not be opened until used in its correct context as a responder receiving a request message with **AmReceiver.receive**. When opened, its queue and queue manager properties are modified to reflect the *ReplyTo* destination specified in the message being received. When first used in this context, the sender service becomes a responder sender service.

clearErrorCodes

Clears the error codes in the AmSender.

```
void clearErrorCodes();
```

close

Closes the AmSender. **close** is overloaded: the policy parameter is optional.

```
void close(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void enableWarnings(boolean warningsOn);
```

warningsOn If set to true, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier for the AmSender. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the sender must perform CCSID conversion of the message before it is sent.

```
int getCCSID();
```

getEncoding

Returns the value used to encode numeric data types for the AmSender. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the sender must convert the encoding of the message before it is sent.

```
int getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmSender.

```
String getName();
```

open

Opens an AmSender service. **open** is overloaded: the policy parameter is optional.

```
void open(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

send

Sends a message to the destination specified by the AmSender. If the AmSender is not open, it will be opened (if this action is specified in the policy options).

send is overloaded: the sendMessage parameter is required, but the others are optional. receivedMessage and responseService are used in request/response messaging, and are mutually exclusive.

```
void send(
    AmMessage  sendMessage,
    AmReceiver  responseService,
    AmMessage  receivedMessage,
    AmPolicy    policy);
```

sendMessage The message object that contains the data to be sent.

responseService

The AmReceiver to be used for receiving any response to the sent message. If omitted, no response can be received.

receivedMessage

The previously received message which is used for correlation with the sent message. If omitted, the sent message is not correlated with any received message.

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

Java AmSender

sendFile

Sends data from a file. To send data from a file, the `sendMessage` and `fileName` parameters are required, but the `policy` is optional. The file data can be received as normal message data by a target application using `AmReceiver.receive`, or used to reconstruct the file with `AmReceiver.receiveFile`.

```
void sendFile(  
    AmMessage sendMessage,  
    String filename,  
    AmPolicy policy);
```

- sendMessage** The message object to use to send the file. This can be used to specify the Correlation ID for example.
- fileName** The name of the file to be sent (input). This can include a directory prefix to define a fully-qualified or relative file name. If the send operation is a physical-mode file transfer, the file name will travel with the message for use with the receive file method (see “receiveFile” on page 402 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.
- policy** The policy to be used. If omitted, the system default policy (name constant: `AMSD_POL`) is used.

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.

Java AmReceiver

responseService

The AmSender to be used for sending any response to the browsed message. If omitted, no response can be sent.

Specify this parameter only when the AMBRW_RECEIVE_CURRENT browse option is used to receive (rather than browse) the message currently under the browse cursor.

selectionMessage

A message object which contains the Correlation ID used to selectively browse a message from the AmReceiver. If omitted, the first available message is browsed. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be reset (see **AmMessage.setCorrelationId** on page 394) before invoking the browse method.

policy

The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

clearErrorCodes

Clears the error codes in the AmReceiver.

```
void clearErrorCodes();
```

close

Closes the AmReceiver. **close** is overloaded: the **policy** parameter is optional.

```
void close(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void enableWarnings(boolean warningsOn);
```

warningsOn If set to true, AmWarningExceptions will be raised for this object.

getDefinitionType

Returns the definition type (service type) for the AmReceiver.

```
int getDefinitionType();
```

The following values can be returned:

```
AMDT_UNDEFINED  
AMDT_TEMP_DYNAMIC  
AMDT_DYNAMIC  
AMDT_PREDEFINED
```

Values other than AMDT_UNDEFINED reflect the properties of the underlying queue object.

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmReceiver.

```
String getName();
```

getQueueName

Returns the queue name of the AmReceiver. This is used to determine the queue name of a permanent dynamic AmReceiver, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **setQueueName**.)

```
String getQueueName();
```

open

Opens an AmReceiver service. **open** is overloaded: the policy parameter is optional.

```
void open(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

receive

Receives a message from the AmReceiver service. **receive** is overloaded: the receiveMessage parameter is required, but the others are optional.

```
void receive(
    AmMessage receiveMessage,
    AmSender responseService,
    AmMessage selectionMessage,
    AmPolicy policy);
```

receiveMessage

The message object that receives the data. The message object is reset implicitly before the receive takes place.

responseService

The AmSender to be used for sending any response to the received message. If omitted, no response can be sent.

selectionMessage

A message object containing the Correlation ID used to selectively receive a message from the AmReceiver. If omitted, the first available message is received. The CCSID, element CCSID and encoding values from the selection message define the target values for data conversion. If target conversion values are required without using the Correlation ID for selection then this can be reset (see **AmMessage.setCorrelationId** on page 394) before invoking the receive method.

policy

The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

Java AmReceiver

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 reset (see **AmMessage.setCorrelationId** on page 394) before invoking the receive method.

policy

The policy to be used. If omitted, the system default policy (constant: `AMSD_POL`) is used.

setQueueName

Sets the queue name of the AmReceiver (when this encapsulates a model queue). This is used to specify the queue name of a recreated permanent dynamic AmReceiver, in order to receive messages in a session subsequent to the one in which it was created. (See also **getQueueName**.)

```
void setQueueName(String queueName);
```

queueName

The queue name to be set in the AmReceiver.

AmDistributionList

An **AmDistributionList** object encapsulates a list of AmSender objects.

clearErrorCodes

Clears the error codes in the AmDistributionList.

```
void clearErrorCodes();
```

close

Closes the AmDistributionList. **close** is overloaded: the *policy* parameter is optional.

```
void close(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void enableWarnings(boolean warningsOn);
```

warningsOn If set to true, AmWarningExceptions will be raised for this object.

getLastErrorStatus

Returns the AmStatus of the last error condition of this object.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmDistributionList object.

```
String getName();
```

getSender

Returns the AmSender in the AmDistributionList object at the index specified. **AmDistributionList.getSenderCount** gets the number of AmSender services in the distribution list.

```
AmSender getSender(int index);
```

index The index of the AmSender in the AmDistributionList, starting at zero.

getSenderCount

Returns the number of AmSender services in the AmDistributionList object.

```
int getSenderCount();
```

open

Opens an AmDistributionList object for each of the destinations in the distribution list. **open** is overloaded: the *policy* parameter is optional.

```
void open(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

Java AmDistributionList

send

Sends a message to each AmSender defined in the AmDistributionList object. **send** is overloaded: the sendMessage parameter is required, but the others are optional.

```
void send(  
    AmMessage  sendMessage,  
    AmReceiver responseService,  
    AmPolicy   policy);
```

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 402 for more details). Note that the file name sent will exactly match the supplied file name; it will not be converted or expanded in any way.

policy

The policy to be used. If omitted, the system default policy (name constant: AMSD_POL) is used.

AmPublisher

An **AmPublisher** object encapsulates an AmSender and provides support for publish requests to a publish/subscribe broker.

clearErrorCodes

Clears the error codes in the AmPublisher.

```
void clearErrorCodes();
```

close

Closes the AmPublisher. **close** is overloaded: the policy parameter is optional.

```
void close(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void enableWarnings(boolean warningsOn);
```

warningsOn If set to true, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier for the AmPublisher. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the publisher must perform CCSID conversion of the message before it is sent.

```
int getCCSID();
```

getEncoding

Returns the value used to encode numeric data types for the AmPublisher. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the publisher must convert the encoding of the message before it is sent.

```
int getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmPublisher.

```
String getName();
```

Java AmPublisher

open

Opens an AmPublisher service. **open** is overloaded: the `policy` parameter is optional.

```
void open(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (AMSD_POL) is used.

publish

Publishes a message using the AmPublisher. **publish** is overloaded: the `pubMessage` parameter is required, but the others are optional.

```
void publish(  
    AmMessage pubMessage,  
    AmReceiver responseService,  
    AmPolicy policy);
```

pubMessage The message object that contains the data to be published.

responseService

The AmReceiver to which the response to the publish request should be sent. Omit it if no response is required. This parameter is mandatory if the policy specifies implicit registration of the publisher.

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmSubscriber

An **AmSubscriber** object encapsulates both an AmSender and an AmReceiver. It provides support for subscribe and unsubscribe requests to a publish/subscribe broker, and for receiving publications from the broker.

clearErrorCodes

Clears the error codes in the AmSubscriber.

```
void clearErrorCodes();
```

close

Closes the AmSubscriber. **close** is overloaded: the policy parameter is optional.

```
void close(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void enableWarnings(boolean warningsOn);
```

warningsOn If set to true, AmWarningExceptions will be raised for this object.

getCCSID

Returns the coded character set identifier for the AmSender in the AmSubscriber. A non-default value reflects the CCSID of a remote system unable to perform CCSID conversion of received messages. In this case the subscriber must perform CCSID conversion of the message before it is sent.

```
int getCCSID();
```

getDefinitionType

Returns the definition type for the AmReceiver in the AmSubscriber.

```
int getDefinitionType();
```

The following values can be returned:

```
AMDT_UNDEFINED
AMDT_TEMP_DYNAMIC
AMDT_DYNAMIC
AMDT_PREDEFINED
```

getEncoding

Returns the value used to encode numeric data types for the AmSender in the AmSubscriber. A non-default value reflects the encoding of a remote system unable to convert the encoding of received messages. In this case the subscriber must convert the encoding of the message before it is sent.

```
int getEncoding();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```


Java AmSubscriber

getName

Returns the name of the AmSubscriber.

```
String getName();
```

getQueueName

Returns the queue name used by the AmSubscriber to receive messages. This is used to determine the queue name of a permanent dynamic AmReceiver in the AmSubscriber, so that it can be recreated with the same queue name in order to receive messages in a subsequent session. (See also **setQueueName**.)

```
String getQueueName();
```

open

Opens an AmSubscriber. **open** is overloaded: the policy parameter is optional.

```
void open(AmPolicy policy);
```

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

receive

Receives a message, normally a publication, using the AmSubscriber. The message data, topic and other elements can be accessed using the message interface methods (see page 389).

receive is overloaded: the pubMessage parameter is required, but the others are optional.

```
void receive(  
    AmMessage pubMessage,  
    AmMessage selectionMessage,  
    AmPolicy policy);
```

pubMessage The message object containing the data that has been published. The message object is reset implicitly before the receive takes place.

selectionMessage

A message object containing the correlation ID used to selectively receive a message from the AmSubscriber. If omitted, the first available message is received.

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

setQueueName

Sets the queue name in the AmReceiver of the AmSubscriber, when this encapsulates a model queue. This is used to specify the queue name of a recreated permanent dynamic AmReceiver, in order to receive messages in a session subsequent to the one in which it was created. (See also **getQueueName**.)

```
void setQueueName(String queueName);
```

queueName

The queue name to be set.

subscribe

Sends a subscribe message to a publish/subscribe broker using the AmSubscriber, to register a subscription. The topic and other elements can be specified using the message interface methods (see page 389) before sending the message.

Publications matching the subscription are sent to the AmReceiver associated with the AmSubscriber. By default, this has the same name as the AmSubscriber, with the addition of the suffix '.RECEIVER'.

subscribe is overloaded: the subMessage parameter is required, but the others are optional.

```
void subscribe(
    AmMessage subMessage,
    AmReceiver responseService,
    AmPolicy policy);
```

subMessage The message object that contains the topic subscription data.

responseService

The AmReceiver to which the response to this subscribe request should be sent. Omit it if no response is required.

This is not the AmReceiver to which publications will be sent by the broker; they are sent to the AmReceiver associated with the AmSubscriber (see above).

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

unsubscribe

Sends an unsubscribe message to a publish/subscribe broker using the AmSubscriber, to deregister a subscription. The topic and other elements can be specified using the message interface methods (see page 389) before sending the message.

unsubscribe is overloaded: the unsubMessage parameter is required, but the others are optional.

```
void unsubscribe(
    AmMessage unsubMessage,
    AmReceiver responseService,
    AmPolicy policy);
```

unsubMessage The message object that contains the topics to which the unsubscribe request applies.

responseService

The AmReceiver to which the response to this unsubscribe request should be sent. Omit it if no response is required.

policy The policy to be used. If omitted, the system default policy (constant: AMSD_POL) is used.

AmPolicy

An **AmPolicy** object encapsulates details of how the AMI processes the message (for instance, the priority and persistence of the message, how errors are handled, and whether transactional processing is used).

clearErrorCodes

Clears the error codes in the AmPolicy.

```
void clearErrorCodes();
```

enableWarnings

Enables AmWarningExceptions; the default value for any AmObject is that AmWarningExceptions are not raised. Note that warning reason codes can be retrieved using **getLastErrorStatus**, even if AmWarningExceptions are disabled.

```
void enableWarnings(boolean warningsOn);
```

warningsOn If set to true, AmWarningExceptions will be raised for this object.

getErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmPolicy object.

```
String getName();
```

getWaitTime

Returns the wait time (in ms) set for this AmPolicy.

```
int getWaitTime();
```

setWaitTime

Sets the wait time for any **receive** using this AmPolicy.

```
void setWaitTime(int waitTime);
```

waitTime The wait time (in ms) to be set in the AmPolicy.

AmConstants

This class provides access to the AMI constants listed in “Appendix B. Constants” on page 509.

For example, to use the constant `AMRC_NONE` (an AMI reason code), specify `AmConstants.AMRC_NONE`.

Note: Not all of the constants available in the C and C++ programming interfaces are available in Java, because they are not all appropriate in this language. For instance, `AmConstants` does not contain `AMB_TRUE` or `AMB_FALSE`, because the Java language has its own true and false constants and these are used by the AMI for Java.

AmElement

An **AmElement** object encapsulates a name/value pair which can be added to an AmMessage object.

AmElement

Constructor for an AmElement object.

```
AmElement(String name, String value);
```

name The name of the element.

value The value of the element.

getName

Returns the name of the AmElement.

```
String getName();
```

getValue

Returns the value of the AmElement.

```
String getValue();
```

getVersion

Returns the version of the AmElement (the default value is AmConstants.AMELEM_VERSION_1).

```
int getVersion();
```

setVersion

Sets the version of the AmElement.

```
void setVersion(int version);
```

version The version of the AmElement that is set. It can take the value AmConstants.AMELEM_VERSION_1 or AmConstants.AMELEM_CURRENT_VERSION.

toString

Returns a String representation of the AmElement.

```
String toString();
```

AmObject

AmObject is a Java Interface. The following classes implement the AmObject interface:

- AmSession
- AmMessage
- AmSender
- AmReceiver
- AmDistributionList
- AmPublisher
- AmSubscriber
- AmPolicy

This allows application programmers to use generic error handling routines.

clearErrorCodes

Clears the error codes in the AmObject.

```
void clearErrorCodes();
```

getLastErrorStatus

Returns the AmStatus of the last error condition.

```
AmStatus getLastErrorStatus();
```

getName

Returns the name of the AmObject.

```
String getName();
```

AmStatus

An **AmStatus** object encapsulates the error status of other AmObjects.

AmStatus

Constructor for an AmStatus object.

```
AmStatus();
```

getCompletionCode

Returns the completion code from the AmStatus object.

```
int getCompletionCode();
```

getReasonCode

Returns the reason code from the AmStatus object.

```
int getReasonCode();
```

getReasonCode2

Returns the secondary reason code from the AmStatus object. (This code is specific to the underlying transport used by the AMI). For MQSeries, the secondary reason code is an MQSeries reason code of type MQRC_XXX.

```
int getReasonCode2();
```

toString

Returns a String representation of the internal state of the AmStatus object.

```
String toString();
```

AmException

AmException is the base Exception class; all other Exceptions inherit from this class.

getClassName

Returns the type of object throwing the Exception.

```
String getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
String getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject getSource();
```

toString

Returns a String representation of the Exception.

```
String toString();
```


AmErrorException

An Exception of type **AmErrorException** is raised when an object experiences an error with a severity level of FAILED (CompletionCode = AMCC_FAILED).

getClassName

Returns the type of object throwing the Exception.

```
String getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
String getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject getSource();
```

toString

Returns a String representation of the Exception.

```
String toString();
```

AmWarningException

An Exception of type **AmWarningException** is raised when an object experiences an error with a severity level of WARNING (CompletionCode = AMCC_WARNING).

getClassName

Returns the type of object throwing the Exception.

```
String getClassName();
```

getCompletionCode

Returns the completion code for the Exception.

```
int getCompletionCode();
```

getMethodName

Returns the name of the method throwing the Exception.

```
String getMethodName();
```

getReasonCode

Returns the reason code for the Exception.

```
int getReasonCode();
```

getSource

Returns the AmObject throwing the Exception.

```
AmObject getSource();
```

toString

Returns a String representation of the Exception.

```
String toString();
```

Java AmWarningException

Part 6. OS/390 Subsystems

Chapter 16. Writing applications for OS/390

subsystems	421
Writing IMS applications using AMI	421
Writing CICS applications using AMI	421
Writing batch applications using AMI	422
Writing RRS-batch applications using AMI	422
RRS availability	422

Chapter 16. Writing applications for OS/390 subsystems

Here is some advice for those of you who want to write AMI applications for the IMS, CICS, batch, and RRS-batch subsystems on OS/390.

Writing IMS applications using AMI

In an IMS application, you establish a syncpoint by using IMS calls such as GU (get unique) to the IOFCB and CHKP (checkpoint). To back out changes since the previous checkpoint, you can use the IMS ROLB (rollback) call. For more information, see the following manuals:

- *IMS/ESA Application Programming: Transaction Manager*
- *IMS/ESA Application Programming: Design Guide*

If other recoverable resources are also involved in the unit of work, the queue manager (in conjunction with the IMS syncpoint manager) participates in a two-phase commit protocol; otherwise, the queue manager performs a single-phase commit process.

All AMI sessions are marked as expired at a syncpoint or rollback (except in a batch-orientated BMP). This is because a different user could initiate the next unit of work and MQSeries security checking is performed when an AMI session or service is opened, not when an AMI object is accessed.

Any subsequent use of a session that has been marked expired (or any object created using that session), will return AMRC_SESSION_EXPIRED. It is the application's responsibility to ensure that all AMI sessions marked as expired are actually deleted.

We recommend that applications explicitly end all AMI sessions (using amSesDelete or amTerminate) before syncpoint, to ensure that any AMI reason codes are correctly reported to the application, and to help ensure that all AMI sessions are deleted.

If an IMS application closes or deletes an AMI session, no implicit syncpoint is taken. If the application closes down normally, any open services are closed and an implicit commit occurs. If the application closes down abnormally, any open services are closed and an implicit backout occurs.

Writing CICS applications using AMI

In a CICS application, you establish a syncpoint by using CICS calls such as EXEC CICS SYNCPOINT. To back out changes to the previous syncpoint you can use the EXEC CICS SYNCPOINT ROLLBACK call. For more information, see the *CICS Application Programming Reference* manual.

If other recoverable resources are also involved in the unit of work, the queue manager (in conjunction with the CICS syncpoint manager) participates in a two-phase commit protocol; otherwise, the queue manager performs a single-phase commit process.

If a CICS application closes or deletes an AMI session, no implicit syncpoint is taken. If the application closes down normally, any open services are closed and an

implicit commit occurs. If the application closes down abnormally, any open services are closed and an implicit backout occurs. Note that file transfer calls are not supported under CICS. If used in a CICS application on OS/390, they return the reason code: AMRC_FILE_TRANSFER_INVALID (144).

If the AMI detects an internal processing error from which no recovery is possible, CICS applications will create a CICS transaction dump, with identifier 'MAMT'. See "First failure symptom report (OS/390)" on page 492 for more details.

Writing batch applications using AMI

In a batch application, you establish a syncpoint by using AMI calls such as amCommit or amSesCommit. To back out changes to the previous syncpoint you can use the amBackout or amSesRollback calls.

Note: If you need to commit or back out updates to resources managed by different resource managers, such as MQSeries and DB2®, within a single unit of work, you should use RRS. For further information, see "Writing RRS-batch applications using AMI".

If a batch application closes or deletes an AMI session, an implicit syncpoint is taken. If the application closes down normally, without first closing or deleting an AMI session, an implicit syncpoint occurs. If the application closes down abnormally, an implicit backout occurs.

Writing RRS-batch applications using AMI

Transaction management and recoverable resource services (RRS) is an OS/390 facility that provides two-phase syncpoint support across participating resource managers. An application can update recoverable resources managed by various OS/390 resource managers such as MQSeries and DB2 and then commit or back out these changes as a single unit of work.

In a RRS-batch application, you establish a syncpoint by using RRS calls such as SRRCMIT. To back out changes to the previous syncpoint you can use the SRRBACK call. For more information, see the *MVS Callable Services for High Level Languages* manual.

RRS availability

If RRS is not active on your OS/390 system, any AMI call which resolves to an MQSeries call will return one of the following AMI reason codes:

AMI reason code	Reason code 2
AMRC_TRANSPORT_ERROR	MQRC_ENVIRONMENT_ERROR
AMRC_BACKOUT_INVALID	NONE
AMRC_COMMIT_INVALID	NONE

If an RRS application closes or deletes an AMI session, no implicit syncpoint is taken. If the application closes down normally, any open services are closed and an implicit commit occurs. If the application closes down abnormally, any open services are closed and an implicit backout occurs.

Part 7. Setting up an AMI installation

Chapter 17. Installation and sample programs	425	Local host and repository files (AS/400, UNIX, and Windows)	450
Prerequisites	425	Default location	450
Disk space	425	Default names	450
Operating environments	425	Overriding the default location and names	450
MQSeries environment	426	Local host file	451
Language compilers	426	Repository file	452
Installation on AIX	428	Local host and repository files (OS/390)	453
Installation	428	Batch, RRS-batch, IMS	453
Manual installation	428	CICS	453
Using amtInstall	428	Local host file	453
Removing the AMI	428	Repository file	454
Setting the runtime environment	429	Repository and local host caches	454
Java programs	429	Generating caches	454
Directory structure (AIX)	430	Using a cache	455
Installation on AS/400	432	Cache generator messages	455
Setting the runtime environment for Java programs	432	The administration tool	457
Directory structure (AS/400)	433	Installation	457
Library QMQMAMI	433	Operation	457
Members of QMQMAMI/H	433	Connecting to MQSeries	458
Members of QMQMAMI/AMTMQSC	433	Using MQSeries Integrator Version 1	458
Members of QMQMAMI/QCSRC	433	Using MQSeries Publish/Subscribe	458
Members of QMQMAMI/QCPPSRC	434	Using MQSeries Integrator Version 2	458
IFS	434	Migrating to MQSeries Integrator V2 from V1 and MQSeries Publish/Subscribe	459
Installation on HP-UX	436	Creating default MQSeries objects	460
Installation	436	The sample programs	461
Manual installation	436	Sample programs for AS/400, UNIX, and Windows	461
Using amtInstall	436	Running the AS/400, UNIX, and Windows sample programs	462
Removing the AMI	436	MQSeries objects	462
Setting the runtime environment	437	Repository and host files	462
Java programs	437	Running the publish/subscribe samples	462
Directory structure (HP-UX)	438	Setting the runtime environment	463
Installation on OS/390	440	Running the C and C++ samples	463
Installation	440	Running the Java samples	463
Setting the runtime environment	440	Running the sample programs (AS/400)	463
Batch and RRS-batch	440	Sample programs for OS/390	464
IMS	440	Running the sample programs (OS/390)	465
CICS	440	Building the sample programs	465
Unicode character conversion	440	MQSeries objects	465
Batch, RRS-batch, IMS	440	Repository and host files	465
CICS	440	Running the publish/subscribe samples	465
Directory structure (OS/390)	441	Setting the runtime environment	466
Installation on Sun Solaris	443	File name input for the file transfer samples	466
Installation	443	Running the batch samples	466
Manual installation	443	Running the CICS samples	467
Using amtInstall	443	Running the IMS samples	467
Removing the AMI	443	Chapter 18. Defining services and policies	469
Setting the runtime environment	444	Services and policies	469
Java programs	444	System provided definitions	470
Directory structure (Solaris)	445	System default objects	470
Installation on Windows	447	Service definitions	472
Installation	447	Service point (sender/receiver)	472
Removing the AMI	447		
Setting the runtime environment	447		
Directory structure (Windows)	448		

Distribution list	474
Subscriber	474
Publisher	474
Policy definitions	475
Initialization attributes	475
General attributes	476
Send attributes	476
Receive attributes	478
Subscribe attributes	479
Publish attributes	480
Chapter 19. Problem determination	481
Using trace (AS/400, UNIX, and Windows)	481
Trace filename and directory	482
Commands on AS/400	482
Commands on UNIX	482
Commands on Windows	483
C++ and Java	484
Example trace	484
Using trace (OS/390)	489
Formatted Trace	489
Control of formatted trace	489
GTF Trace	490
Control of GTF Trace	490
When your AMI program fails	492
Reason Codes	492
First failure symptom report (AS/400, UNIX, and Windows)	492
First failure symptom report (OS/390)	492
Other sources of information	493
Common causes of problems	493

Chapter 17. Installation and sample programs

The Application Messaging Interface is available for the AIX, AS/400, HP-UX, OS/390, Sun Solaris, Windows NT, and Windows 98 platforms.

This chapter contains:

- "Prerequisites"
- "Installation on AIX" on page 428
- "Installation on AS/400" on page 432
- "Installation on HP-UX" on page 436
- "Installation on OS/390" on page 440
- "Installation on Sun Solaris" on page 443
- "Installation on Windows" on page 447
- "Local host and repository files (AS/400, UNIX, and Windows)" on page 450
- "The administration tool" on page 457
- "Connecting to MQSeries" on page 458
- "The sample programs" on page 461

Prerequisites

Before you install the AMI, ensure that your system has sufficient disk space, and has the software listed in the following sections.

Disk space

Disk space requirements:

AIX	15.0 MB
AS/400	30.0 MB
HP-UX	12.7 MB
OS/390	Not applicable (AMI installed as part of MQSeries for OS/390)
Sun Solaris	11.2 MB
Windows	10.9 MB (without AMI Administration Tool) 22.4 MB (with AMI Administration Tool)

Operating environments

The AMI runs under the following operating systems:

AIX	V4.3
AS/400	V4R4 or later
HP-UX	V11.0
OS/390	V2R6 or later, with Language Environment CICS 4.1 or later, with Language Environment IMS V5.1 or later, with Language Environment
Sun Solaris	V2.6 and V7
Windows	Windows NT V4 and Windows 98

Prerequisites

MQSeries environment

You can run the AMI in an MQSeries server or client environment.

To run the AMI in an MQSeries server environment, you need at least one of the following installed on your system:

- MQSeries for AIX Version 5.1 or later
- MQSeries for AS/400 Version 5.1 or later
- MQSeries for HP-UX Version 5.1 or later
- MQSeries for OS/390 Version 5.2
- MQSeries for Sun Solaris Version 5.1 or later
- MQSeries for Windows NT Version 5.1 or later

To run the AMI in an MQSeries client environment, you need at least one of the following installed on your system:

- MQSeries client for AIX Version 5.1 or later
- MQSeries client for HP-UX Version 5.1 or later
- MQSeries client for Sun Solaris Version 5.1 or later
- MQSeries client for Windows NT Version 5.1 or later
- MQSeries client for Windows 98 Version 5.1 or later

The MQSeries client requires access to at least one supporting MQSeries server.

Language compilers

The following language compilers for C, COBOL, C++, and Java are supported:

AIX	VisualAge® C++ 5.0 JDK™ 1.1.7 and later
AS/400	AS/400 Developer Kit for Java (5769JV1) ILE C for AS/400 (5769CX2) ILE C++ for AS/400 (5799GDW) VisualAge C++ for OS/400 (5716CX4)
HP-UX	HP aC++ B3910B A.03.10 HP aC++ B3910B A.03.04 (970930) Support library JDK 1.1.7 and later
OS/390	OS/390 C/C++ Version 2 Release 6 and later IBM COBOL for OS/390 & VM Version 2 Release 1 and later IBM COBOL for MVS & VM Version 1 Release 2 and later
Sun Solaris	Workshop Compiler 4.2 (with Solaris 2.6) Workshop Compiler 5.0 (with Solaris 7) JDK 1.1.7 and later
Windows	Microsoft Visual C++ 6 JDK 1.1.7 and later

Next step

Now go to one of the following to start the installation procedure:

- "Installation on AIX" on page 428
- "Installation on AS/400" on page 432
- "Installation on HP-UX" on page 436
- "Installation on OS/390" on page 440
- "Installation on Sun Solaris" on page 443
- "Installation on Windows" on page 447

Installation on AIX

The AMI package for AIX comes as a compressed archive file, `ma0f_ax.tar.Z`. Uncompress and restore it as follows:

1. Log in as root
2. Store `ma0f_ax.tar.Z` in `/tmp`
3. Execute `uncompress -fv /tmp/ma0f_ax.tar.Z`
4. Execute `tar -xvf /tmp/ma0f_ax.tar`
5. Execute `rm /tmp/ma0f_ax.tar`

This creates the following files:

amt100.tar	A standard tar file containing the AMI files
amtInstall	A script file to aid AMI installation
amtRemove	A script file to aid AMI removal
readme	A file containing any product and information updates that have become available since this documentation was produced

Installation

Installation can be carried out manually, or using the **amtInstall** utility.

Manual installation

Restore the tar file `amt100.tar`. This should be done under the base MQSeries directory `/usr/mqm`, so that the AMI tar file restores to a directory structure consistent with MQSeries. This operation usually requires root access. Existing files will be overwritten. (Note that the location `/usr/mqm/` is consistent with MQSeries Version 5.1, which is the prerequisite for the AMI).

Using amtInstall

1. Log in as root
2. Execute `amtInstall <directory>`

where `<directory>` is the directory containing the `amt100.tar` file.

The **amtInstall** utility will unpack the tar file into the correct location and provide the necessary links for your environment. Existing files will be overwritten.

Note: All files and directories created must be accessible to all AMI users. These files are listed in "Directory structure (AIX)" on page 430.

Removing the AMI

Run the **amtRemove** utility to remove all the files that were created by **amtInstall**.

Setting the runtime environment

Make sure that the location of the AMI runtime binary files is added to your PATH environment variable. For example:

```
export PATH=$PATH:/usr/mqm/lib:
```

Note: The previous step is not needed if you used the **amtInstall** utility.

In addition, for the samples:

```
export PATH=$PATH:/usr/mqm/amt/samp/C/bin:/usr/mqm/amt/samp/Cpp/bin:
```

Java programs

When running Java, there are some additional steps.

The AMI classes must be contained in the CLASSPATH, for example:

```
export CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/com.ibm.mq.amt.jar:
```

In addition, for the samples:

```
export CLASSPATH=$CLASSPATH:/usr/mqm/amt/samp/java/bin  
/com.ibm.mq.amt.samples.jar:
```

Also, to load the AMI library for Java:

```
export LIBPATH=$LIBPATH:/usr/mqm/lib:
```

Next step

Now go to “Local host and repository files (AS/400, UNIX, and Windows)” on page 450 to continue the installation procedure.

Installation on AIX

Directory structure (AIX)

The AMI tar file contains:

```
/amt/amtsdfts.tst : MQSeries mqsc command file to create default MQSeries
objects required by the AMI

/amt/amthost.xml : Sample AMI XML file used as the default host file

/amt/amt.dtd : AMI Document Type Definition file on which the AMI
repository is based

/amt/inc
  amtc.h : The C header file for the AMI
  amtcpp.hpp : The C++ header file for the AMI
  oamasami.h: The C header file for the OAMAS AMI subset

/amt/ipla : The International Program License Agreement file
/amt/li : The License Information file

/java/lib
  com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java

/lib
  libamt.a : The main AMI library
  libamt_r.a : The main AMI threaded library
  libamtXML310.a : The AMI XML parsing library
  libamtXML310_r.a : The AMI threaded XML parsing library
  libamtCpp.a : The AMI C++ library
  libamtCpp_r.a : The AMI C++ threaded library
  libamtJava.so: The AMI JNI library
  libamtICUUC140.a : The AMI codepage translation library
  libamtICUUC140_r.a : The AMI codepage translation threaded library
  libamtICUDATA.a : The AMI codepage translation data library.
  amtcqm : Dynamic binding stub for Server library
  amtcqm_r : Dynamic binding stub for MQSeries Server threaded library
  amtcqic : Dynamic binding stub for MQSeries Client library
  amtcqic_r : Dynamic binding stub for MQSeries Client threaded library

/amt/samp
  amtsamp.tst : MQSeries mqsc command file to create MQSeries objects
required by AMI samples
  amt.xml : Sample AMI XML repository for use with the AMI samples

/amt/samp/C
  amtsosnd.c : C source for object-level send and forget sample
  amtsorcv.c : C source for object-level receiver sample
  amtsoclt.c : C source for object-level client sample
  amtsosvr.c : C source for object-level server sample
  amtsopub.c : C source for object-level publisher sample
  amtsosub.c : C source for object-level subscriber sample
  amtsofsn.c : C source for object-level send file sample
  amtsofrc.c : C source for object-level receive file sample
  amtsosgs.c : C source for object-level send group sample
  amtsosgr.c : C source for object-level receive group sample
  amtshsnd.c : C source for high-level send and forget sample
  amtshrcv.c : C source for high-level receiver sample
  amtshcvt.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
SendAndForget.cpp : C++ source for send and forget sample
Receiver.cpp : C++ source for receiver sample
Client.cpp : C++ source for client sample
Server.cpp : C++ source for server sample
Publisher.cpp : C++ source for publisher sample
Subscriber.cpp : C++ source for subscriber sample
ReceiveFile.cpp : C++ source for receive file sample
SendFile.cpp : C++ source for send file sample

/amt/samp/Cpp/bin
SendAndForget : C++ send and forget sample program
Receiver : C++ receiver sample program
Client : C++ client sample program
Server : C++ server sample program
Publisher : C++ publisher sample program
Subscriber : C++ subscriber sample program
ReceiveFile : C++ source for receive file sample
SendFile : C++ source for send file sample

/amt/samp/java
SendAndForget.java : Java source for send and forget sample
Receiver.java : Java source for receiver sample
Client.java : Java source for client sample
Server.java : Java source for server sample
Publisher.java : Java source for publisher sample
Subscriber.java : Java source for subscriber sample
ReceiveFile.java : Java source for receive file sample
SendFile.java : Java source for send file sample

/amt/samp/java/bin
com.ibm.mq.amt.samples.jar : The jar file containing the AMI
samples class files for Java

```


Installation on 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:
`CRSAVF FILE(QGPL/MA0F)`
4. Transfer `ma0f_400.sav` into this save file as a binary image.
If you use FTP to do this, the put command should be similar to:
`PUT C:\TEMP\MA0F_400.SAV QGPL/MA0F`
5. Install the MQSeries for AS/400 AMI, product Id 5724A23, using RSTLICPGM:
`RSTLICPGM LICPGM(5724A23) DEV(*SAVF) SAVF(QGPL/MA0F)`
6. Delete the save file created in Step 3:
`DLTF FILE(QGPL/MA0F)`

To remove the AMI package from the AS/400, use DLTLICPGM:
`DLTLICPGM LICPGM(5724A23)`

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 ADDLIB command.
- Ensure that the AMI classes are contained in the CLASSPATH.
Use the WRKENVVAR command to determine whether a CLASSPATH exists.
If a CLASSPATH environment variable does not exist, use the ADDENVVAR command to create one. For example:
`ADDENVVAR ENVVAR(CLASSPATH) VALUE('/QIBM/ProdData/mqm/amt/java/lib/com.ibm.mq.amt.jar')`

If a CLASSPATH environment variable already exists, use the CHGENVVAR command to add the following to it:

```
:/QIBM/ProdData/mqm/amt/java/lib/com.ibm.mq.amt.jar
```

To use the supplied samples, include the following in the CLASSPATH:

```
:/QIBM/ProdData/mqm/amt/samp/java/bin/com.ibm.mq.amt.samples.jar
```

Note: Each CLASSPATH entry must be separated by a colon.

Next step

Now go to “Local host and repository files (AS/400, UNIX, and Windows)” on page 450 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
AMTJAVA	*SRVPGM	: The AMI JNI library
AMTXML	*SRVPGM	: The main AMI XML parsing library
AMTMSG	*MSGF	: AMT message file
QAMT0050	*PRDDFN	: Product definition file
QAMT0029	*PRDL0D	: Language Product Load file
QAMT0050	*PRDL0D	: Code Product Load file
H	*FILE	: AMI header files
AMTMQSC	*FILE	: MQSC command files
QXMLMSG	*MSGF	: XML message file
AMTSHCLT	*PGM	: C high-level client sample program
AMTSHFRC	*PGM	: C high-level receive file sample program
AMTSHFSN	*PGM	: C high-level send file sample program
AMTSH PUB	*PGM	: C high-level subscriber sample program
AMTSHRCV	*PGM	: C high-level receiver sample program
AMTSHSND	*PGM	: C high-level send and forget sample program
AMTSHSUB	*PGM	: C high-level subscriber sample program
AMTSHSVR	*PGM	: C high-level server sample program
AMTSOCLT	*PGM	: C object-level client sample program
AMTSOFRC	*PGM	: C object-level receive file sample program
AMTSOFSN	*PGM	: C object-level send file sample program
AMTSOPUB	*PGM	: C object-level publisher sample program
AMTSORCV	*PGM	: C object-level receiver sample program
AMTSOSGR	*PGM	: C object-level receive group sample program
AMTSOSGS	*PGM	: C object-level send group sample program
AMTSOSND	*PGM	: C object-level send and forget sample program
AMTSOSUB	*PGM	: C object-level subscriber sample program
AMTSOSVR	*PGM	: C object-level server sample program
CLIENT	*PGM	: C++ client sample program
PUBLISHER	*PGM	: C++ publisher sample program
RECEIVER	*PGM	: C++ receiver sample program
RCVFILE	*PGM	: C++ receive file sample program
SENDFORGET	*PGM	: C++ send and forget sample program
SENDFILE	*PGM	: C++ send file sample program
SERVER	*PGM	: C++ server sample program
SUBSCRIBER	*PGM	: C++ subscriber sample program
QCSRC	*FILE	: C sample files
QCPPSRC	*FILE	: C++ samples
AMTIOX1C	*PGM	: Installation Exit Program
AMTIOX0C	*PGM	: Installation Exit Program

Members of QMQMAMI/H

AMTC	: The C header file for AMI
AMTCPP	: The C++ header file for AMI

Note: The members of the file H are copies of the AMI header files in IFS.

Members of QMQMAMI/AMTMQSC

AMTSDFTS	: MQSeries mqsc command file to create default MQSeries objects required by the AMI
AMTSAMP	: MQSeries mqsc command file to create MQSeries objects required by AMI samples

Members of QMQMAMI/QCSRC

amtshclt	: C source for high-level client sample program
amtshfrc	: C source for high-level receive file sample program
amtshfsn	: C source for high-level send file sample program

Installation on AS/400

```
amtshpub      : C source for high-level subscriber sample program
amtshrcv      : C source for high-level receiver sample program
amtshsnd      : C source for high-level send and forget sample program
amtshsub      : C source for high-level subscriber sample program
amtshsvr      : C source for high-level server sample program
amtsoclt      : C source for object-level client sample program
amtsofrc      : C source for object-level receive file sample program
amtsofsn      : C source for object-level send file sample program
amtsopub      : C source for object-level publisher sample program
amtsorcv      : C source for object-level receiver sample program
amtsosgr      : C source for object-level receive group sample program
amtsosgs      : C source for object-level send group sample program
amtsosnd      : C source for object-level send and forget sample program
amtsosub      : C source for object-level subscriber sample program
amtsosvr      : C source for object-level server sample program
```

Note: The members of the file QCSRC are copies of the C source files for sample programs in IFS.

Members of QMQMAMI/QCPPSRC

```
Client        : C++ source for client sample
Publisher     : C++ source for publisher sample
Receiver      : C++ source for receiver sample
RcvFile       : C++ source for receive file sample
SendForget    : C++ source for send and forget sample
SendFile      : C++ source for send file sample
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.

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
inc      : AMI header files
ipla     : The International Program License Agreement file
li       : The License Information file

/QIBM/ProdData/mqm/amt/inc
amtc.h   : The C header file for AMI
amtcpp.hpp : The C++ header file for AMI

/QIBM/ProdData/mqm/amt/samp
amtsamp.tst : MQSeries mqsc command file to create MQSeries objects
              required by AMI samples
amt.xml     : Sample AMI XML repository for use with the AMI samples

/QIBM/ProdData/mqm/amt/samp/C
amtshclt.c : C source for high-level client sample program
amtshfrc.c : C source for high-level receive file sample program
amtshfsn.c : C source for high-level send file sample program
amtshpub.c : C source for high-level subscriber sample program
amtshrcv.c : C source for high-level receiver sample program
amtshsnd.c : C source for high-level send and forget sample program
amtshsub.c : C source for high-level subscriber sample program
amtshsvr.c : C source for high-level server sample program
amtsoclt.c : C source for object-level client sample program
amtsofrc.c : C source for object-level receive file sample program
amtsofsn.c : C source for object-level send file sample program
amtsopub.c : C source for object-level publisher sample program
amtsorcv.c : C source for object-level receiver sample program
amtsosgr.c : C source for object-level receive group sample program
amtsosgs.c : C source for object-level send group sample program
```

Installation on AS/400

amtsosnd.c : C source for object-level send and forget sample program
amtsosub.c : C source for object-level subscriber sample program
amtsosvr.c : C source for object-level server sample program

/QIBM/ProdData/mqm/amt/samp/Cpp

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
ReceiveFile.java : Java source for receive file sample
SendandForget.java : Java source for send and forget sample
SendFile.java : Java source for send file sample
Server.java : Java source for server sample
Subscriber.java : Java source for subscriber sample

/QIBM/ProdData/mqm/amt/samp/java/bin

com.ibm.mq.amt.samples.jar : The jar file containing the AMI
samples class files for Java

/QIBM/ProdData/mqm/amt/java/lib

com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java

/QIBM/UserData/mqm/amt

amt.dtd : Document Type Definition file on which the AMI repository is based
amthost.xml : Sample AMI XML file used as the default host file

Installation on HP-UX

The AMI package for HP-UX comes as a compressed archive file, `ma0f_hp.tar.Z`. Uncompress and restore it as follows:

1. Log in as root
2. Store `ma0f_hp.tar.Z` in `/tmp`
3. Execute `uncompress -fv /tmp/ma0f_hp.tar.Z`
4. Execute `tar -xvf /tmp/ma0f_hp.tar`
5. Execute `rm /tmp/ma0f_hp.tar`

This creates the following files:

amt100.tar	A standard tar file containing the AMI files
amtInstall	A script file to aid AMI installation
amtRemove	A script file to aid AMI removal
readme	A file containing any product and information updates that have become available since this documentation was produced

Installation

Installation can be carried out manually, or using the **amtInstall** utility.

Manual installation

Restore the tar file `amt100.tar`. Do this under the base MQSeries directory `/opt/mqm`, so that the AMI tar file restores to a directory structure consistent with MQSeries. This operation usually requires root access. Existing files will be overwritten.

Using amtInstall

1. Log in as root
2. Execute `amtInstall <directory>`

where `<directory>` is the directory containing the `amt100.tar` file.

The **amtInstall** utility will unpack the tar file into the correct location and provide all the necessary links for your environment. Existing files will be overwritten.

Note: All files and directories created must be accessible to all AMI users. These files are listed in “Directory structure (HP-UX)” on page 438.

Removing the AMI

Run the **amtRemove** utility to remove all the files that were created by **amtInstall**.

Setting the runtime environment

Make sure the location of the AMI runtime binary files is added to your PATH environment variable. For example:

```
export PATH=$PATH:/opt/mqm/lib:
```

Note: The previous step is not needed if you used the **amtInstall** utility.

In addition, for the samples:

```
export PATH=$PATH:/opt/mqm/amt/samp/C/bin:/opt/mqm/amt/samp/Cpp/bin:
```

Java programs

When running Java, there are some additional steps.

The AMI classes must be contained in the CLASSPATH, for example:

```
export CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/com.ibm.mq.ami.jar:
```

In addition, for the samples:

```
export CLASSPATH=$CLASSPATH:/opt/mqm/amt/samp/java/bin
/com.ibm.mq.ami.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 450 to continue the installation procedure.

Installation on HP-UX

Directory structure (HP-UX)

The AMI tar file contains:

```
/amt/amtsdfts.tst : MQSeries mqsc command file to create default MQSeries
  objects required by the AMI

/amt/amthost.xml : Sample AMI XML file used as the default host file

/amt/amt.dtd : AMI Document Type Definition file on which the AMI
  repository is based

/amt/inc
  amtc.h : The C header file for the AMI
  amtcpp.hpp : The C++ header file for the AMI
  oamasami.h : The C header file for the OAMAS AMI subset

/amt/ipla : The International Program License Agreement file
/amt/li : The License Information file

/java/lib
  com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java

/lib
  libamt_r.sl : The main AMI threaded library
  libamtXML310_r.sl : The AMI threaded XML parsing library
  libamtCpp_r.sl : The AMI C++ threaded library
  libamtJava.sl : The AMI JNI library
  libamtICUUC140_r.sl : The AMI codepage translation threaded library
  libamtICUDATA.sl : The AMI codepage translation data library.
  amtcmqm_r : Dynamic binding stub for MQSeries Server threaded library
  amtcmqic_r : Dynamic binding stub for MQSeries Client threaded library

/amt/samp
  amtsamp.tst : MQSeries mqsc command file to create MQSeries objects
    required by AMI samples
  amt.xml : Sample AMI XML repository for use with the AMI samples

/amt/samp/C
  amtsosnd.c : C source for object-level send and forget sample
  amtsorcv.c : C source for object-level receiver sample
  amtsoclt.c : C source for object-level client sample
  amtsosvr.c : C source for object-level server sample
  amtsopub.c : C source for object-level publisher sample
  amtsosub.c : C source for object-level subscriber sample
  amtsofsn.c : C source for object-level send file sample
  amtsofrc.c : C source for object-level receive file sample
  amtsosgs.c : C source for object-level send group sample
  amtsosgr.c : C source for object-level receive group sample
  amtshsnd.c : C source for high-level send and forget sample
  amtshrcv.c : C source for high-level receiver sample
  amtshcvt.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
SendAndForget.cpp : C++ source for send and forget sample
Receiver.cpp : C++ source for receiver sample
Client.cpp : C++ source for client sample
Server.cpp : C++ source for server sample
Publisher.cpp : C++ source for publisher sample
Subscriber.cpp : C++ source for subscriber sample
ReceiveFile.cpp : C++ source for receive file sample
SendFile.cpp : C++ source for send file sample

/amt/samp/Cpp/bin
SendAndForget : C++ send and forget sample program
Receiver : C++ receiver sample program
Client : C++ client sample program
Server : C++ server sample program
Publisher : C++ publisher sample program
Subscriber : C++ subscriber sample program
ReceiveFile : C++ source for receive file sample
SendFile : C++ source for send file sample

/amt/samp/java
SendAndForget.java : Java source for send and forget sample
Receiver.java : Java source for receiver sample
Client.java : Java source for client sample
Server.java : Java source for server sample
Publisher.java : Java source for publisher sample
Subscriber.java : Java source for subscriber sample
ReceiveFile.java : Java source for receive file sample
SendFile.java : Java source for send file sample

/amt/samp/java/bin
com.ibm.mq.amt.samples.jar : The jar file containing the AMI
samples class files for Java

```


Installation on OS/390

The AMI is installed automatically with MQSeries for OS/390 Version 5.2.

Installation

The files and directories created are listed in “Directory structure (OS/390)” on page 441.

Setting the runtime environment

Batch and RRS-batch

Make sure that the location of the AMI runtime library is added to your JCL STEPLIB concatenation.

IMS

Make sure that the location of the AMI runtime library is added to your IMS message processing region JCL STEPLIB concatenation.

CICS

Make sure that the location of the AMI runtime library is added to your region’s DFHRPL concatenation, and the AMI library is defined in your CICS CSD. A sample CSD script, `inhlq.SCSQPROC(AMTCSD10)`, is supplied to help define the AMI library to CICS.

Unicode character conversion

If your OS/390 installation predates OS/390 V2 R9, applications that use the AMI publish subscribe calls, message element calls, and file transfer calls may need to perform some extra configuration. This configuration enables the Language Environment support for Unicode character conversion. With OS/390 V2 R9, the Unicode conversion tables were replaced with direct Unicode converters, enabling higher performance and removing the need for this extra configuration. Refer to the *OS/390 V2R9.0 C/C++ Compiler and Run-Time Migration Guide* for more details.

Batch, RRS-batch, IMS

If your Language Environment is installed in a non-default location, you will need to set the environment variable `_ICONV_UCS2_PREFIX` to specify the value of your installation prefix before running your AMI application. This ensures that the AMI has access to Unicode character conversion tables. See the *OS/390 C/C++ Programming Guide* for examples of setting this environment variable.

CICS

OS/390 releases before OS/390 V2 R9 do not support Unicode character conversions under CICS. This makes it impossible to use AMI publish subscribe and message element support with earlier versions of OS/390.

OS/390 V2 R9 is required to enable AMI publish subscribe or message element support under CICS.

Next step

Now go to “Local host and repository files (OS/390)” on page 453 to continue the installation procedure.

Directory structure (OS/390)

On OS/390 platforms the directory structure contains the following (where 'hlq' is the high-level qualifier of the AMI installation):

```

hlq.SCSQLOAD
  AMTBL10 : The main AMI library (batch)
  AMTCL10 : The main AMI library (CICS)
  AMTIL10 : The main AMI library (IMS)
  AMTRL10 : The main AMI library (RRS-batch)
  AMTBS10 : Stub to build COBOL applications (batch)
  AMTCS10 : Stub to build COBOL applications (CICS)
  AMTIS10 : Stub to build COBOL applications (IMS)
  AMTRS10 : Stub to build COBOL applications (RRS-batch)
  AMTASM10 : Repository cache generator

hlq.SCSQANLE
  AMTMSG10 : US English messages
  AMTMSG10 : US English messages

hlq.SCSQANLU
  AMTMSG10 : Uppercase US English messages
  AMTMSU10 : Uppercase US English messages

hlq.SCSQANLK
  AMTMSG10 : Kanji messages
  AMTMSK10 : Kanji messages

hlq.SCSQANLC
  AMTMSG10 : Chinese messages
  AMTMSC10 : Chinese messages

hlq.SCSQC370
  AMTC : The C header file for the AMI

hlq.SCSQCOBC
  AMTELEML : COBOL copybook for the AMELEM structure
  AMTELEMV : COBOL copybook for the AMELEM structure, with default values
  AMTV : The main COBOL copybook for the AMI

hlq.SCSQPROC
  AMT : Sample AMI XML repository for use with the AMI samples.
  AMTCSD10 : CICS definitions for the AMI library.
  AMTHOST : Sample AMI XML file for use as the default host file (UTF-8).
  AMTHOST2 : Sample AMI XML file for use as the default host file
             (EBCDIC 1047).
  AMTSDFTS : 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.

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.SCSQC OBS (COBOL samples for Batch, RRS, CICS, and IMS)
  AMTVHSND : COBOL source for high-level send and forget sample
  AMTVHRCV : COBOL source for high-level receiver sample
  AMTVHCLT : COBOL source for high-level client sample
  AMTVHsvR : COBOL source for high-level server sample
  AMTVHPUB : COBOL source for high-level publisher sample
  AMTVHsUB : COBOL source for high-level subscriber sample
  AMTVHFSN : COBOL source for high-level group send file transfer sample
  AMTVHFRC : COBOL source for high-level group receive file transfer sample

```

Installation on OS/390

AMTVOSND : COBOL source for object-level send and forget sample
AMTVORCV : COBOL source for object-level receiver sample
AMTVOCLT : COBOL source for object-level client sample
AMTVOSVR : COBOL source for object-level server sample
AMTVOPUB : COBOL source for object-level publisher sample
AMTVOSUB : COBOL source for object-level subscriber sample
AMTVOSGS : COBOL source for object-level group send sample
AMTVOSGR : COBOL source for object-level group receive sample
AMTVOFSN : COBOL source for object-level send file transfer sample
AMTVOFRC : COBOL source for object-level receive file transfer sample

hlq.SCSQC37S (C samples for Batch, RRS, CICS, and IMS)

AMTSHSND : C source for high-level send and forget sample
AMTSHRCV : C source for high-level receiver sample
AMTSHCLT : C source for high-level client sample
AMTSHSVR : C source for high-level server sample
AMTSH PUB : C source for high-level publisher sample
AMTSHSUB : C source for high-level subscriber sample
AMTSHFSN : C source for high-level group send file transfer sample
AMTSHFRC : C source for high-level group receive file transfer sample
AMTSOSND : C source for object-level send and forget sample
AMTSORCV : C source for object-level receiver sample
AMTSOCLT : C source for object-level client sample
AMTSOSVR : C source for object-level server sample
AMTSOPUB : C source for object-level publisher sample
AMTSOSUB : C source for object-level subscriber sample
AMTSOSGS : C source for object-level group send sample
AMTSOSGR : C source for object-level group receive sample
AMTSOFSN : C source for object-level send file transfer sample
AMTSOFRC : C source for object-level receive file transfer sample

Installation on Sun Solaris

The AMI package for Sun Solaris comes as a compressed archive file, `ma0f_sol.tar.Z`. Uncompress and restore it as follows:

1. Log in as root
2. Store `ma0f_sol.tar.Z` in `/tmp`
3. Execute `uncompress -fv /tmp/ma0f_sol.tar.Z`
4. Execute `tar -xvf /tmp/ma0f_sol.tar`
5. Execute `rm /tmp/ma0f_sol.tar`

This creates the following files:

amt100.tar	A standard tar file containing the AMI files
amtInstall	A script file to aid AMI 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 `/opt/mqm`, so that the AMI tar file restores to a directory structure consistent with MQSeries. This operation usually requires root access. Existing files will be overwritten.

Using amtInstall

1. Log in as root
2. Execute `amtInstall <directory>`

where `<directory>` is the directory containing the `amt100.tar` file.

The **amtInstall** utility will unpack the tar file into the correct location and provide the necessary links for your environment. Existing files will be overwritten.

Note: All files and directories created must be accessible to all AMI users. These files are listed in “Directory structure (Solaris)” on page 445.

Removing the AMI

Run the **amtRemove** utility to remove all the files that were created by **amtInstall**.

Installation on Sun Solaris

Setting the runtime environment

Make sure that the location of the AMI runtime binary files is added to your PATH environment variable. For example:

```
export PATH=$PATH:/opt/mqm/lib:
```

Note: The previous step is not needed if you used the **amtInstall** utility.

In addition, for the samples:

```
export PATH=$PATH:/opt/mqm/amt/samp/C/bin:/opt/mqm/amt/samp/Cpp/bin:
```

Java programs

When running Java, there are some additional steps.

The AMI classes must be contained in the CLASSPATH, for example:

```
export CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/com.ibm.mq.amt.jar:
```

In addition, for the samples:

```
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 450 to continue the installation procedure.

Directory structure (Solaris)

The AMI tar file contains:

```

/amt/amtsdfts.tst : MQSeries mqsc command file to create default MQSeries
  objects required by the AMI

/amt/amthost.xml : Sample AMI XML file used as the default host file

/amt/amt.dtd : AMI Document Type Definition file on which the AMI
  repository is based

/amt/inc
  amtc.h : The C header file for the AMI
  amtcpp.hpp : The C++ header file for the AMI
  oamasami.h : The C header file for the OAMAS AMI subset

/amt/ipla : The International Program License Agreement file
/amt/li : The License Information file

/java/lib
  com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java

/lib
  libamt.so : The main AMI library
  libamtXML310.so : The AMI XML parsing library
  libamtCpp.so : The AMI C++ library
  libamtJava.so: The AMI JNI library
  libamtICUUC140.so : The AMI codepage translation library
  libamtICUDATA.so : The AMI codepage translation data library
  amtcmqm : Dynamic binding stub for MQSeries Server library
  amtcmqic : Dynamic binding stub for MQSeries Client library

/amt/samp
  amtsamp.tst : MQSeries mqsc command file to create MQSeries objects
    required by AMI samples
  amt.xml : Sample AMI XML repository for use with the AMI samples

/amt/samp/C
  amtsosnd.c : C source for object-level send and forget sample
  amtsorcv.c : C source for object-level receiver sample
  amtsoclt.c : C source for object-level client sample
  amtsosvr.c : C source for object-level server sample
  amtsopub.c : C source for object-level publisher sample
  amtsosub.c : C source for object-level subscriber sample
  amtsofsn.c : C source for object-level send file sample
  amtsofrc.c : C source for object-level receive file sample
  amtsosgs.c : C source for object-level send group sample
  amtsosgr.c : C source for object-level receive group sample
  amtshsnd.c : C source for high-level send and forget sample
  amtshrcv.c : C source for high-level receiver sample
  amtshclt.c : C source for high-level client sample
  amtshsvr.c : C source for high-level server sample
  amtshpub.c : C source for high-level publisher sample
  amtshsub.c : C source for high-level subscriber sample
  amtshfsn.c : C source for high-level send file sample
  amtshfrc.c : C source for high-level receive file sample

/amt/samp/C/bin
  amtsosnd : C object-level send and forget sample program
  amtsorcv : C object-level receiver sample program
  amtsoclt : C object-level client sample program
  amtsosvr : C object-level server sample program
  amtsopub : C object-level publisher sample program
  amtsosub : C object-level subscriber sample program
  amtsofsn : C object-level send file sample program
  amtsofrc : C object-level receive file sample program

```

Installation on Sun Solaris

```
amtsofrc : C object-level receive file sample program
amtsosgs : C object-level send group sample program
amtsosgr : C object-level receive group sample program
amtshsnd : C high-level send and forget sample program
amtshrcv : C high-level receiver sample program
amtshclt : C high-level client sample program
amtshsvr : C high-level server sample program
amtshpub : C high-level publisher sample program
amtshsub : C high-level subscriber sample program
amtshfsn : C high-level send file sample program
amtshfrc : C high-level receive file sample program

/amt/samp/Cpp
SendAndForget.cpp : C++ source for send and forget sample
Receiver.cpp : C++ source for receiver sample
Client.cpp : C++ source for client sample
Server.cpp : C++ source for server sample
Publisher.cpp : C++ source for publisher sample
Subscriber.cpp : C++ source for subscriber sample
ReceiveFile.cpp : C++ source for receive file sample
SendFile.cpp : C++ source for send file sample

/amt/samp/Cpp/bin
SendAndForget : C++ send and forget sample program
Receiver : C++ receiver sample program
Client : C++ client sample program
Server : C++ server sample program
Publisher : C++ publisher sample program
Subscriber : C++ subscriber sample program
ReceiveFile : C++ source for receive file sample
SendFile : C++ source for send file sample

/amt/samp/java
SendAndForget.java : Java source for send and forget sample
Receiver.java : Java source for receiver sample
Client.java : Java source for client sample
Server.java : Java source for server sample
Publisher.java : Java source for publisher sample
Subscriber.java : Java source for subscriber sample
ReceiveFile.java : Java source for receive file sample
SendFile.java : Java source for send file sample

/amt/samp/java/bin
com.ibm.mq.amt.samples.jar : The jar file containing the AMI
samples class files for Java
```

Installation on Windows

The AMI package for Windows 98 and Windows NT comes as a zip file, `ma0f_nt.zip`. Once unzipped it comprises:

readme

A file containing any product and information updates that have become available since this documentation was produced

setup InstallShield installation program for MQSeries AMI

In addition, it contains files used by the **setup** program.

Installation

1. Create an empty directory called `tmp` and make it current.
2. Store the `ma0f_nt.zip` file in this directory.
3. Uncompress it into `tmp` using Info-ZIP's UnZip program (or other unzip program).
4. Run **setup**.
5. Delete the `tmp` directory.

The files and directories created are listed in "Directory structure (Windows)" on page 448.

Removing the AMI

To uninstall the Application Messaging Interface, use the Add/Remove Programs control panel.

Note: You **must** remove the AMI entries from the CLASSPATH (for instance, `C:\MQSeries\java\lib\com.ibm.mq.amt.jar`; and `C:\MQSeries\amt\samples\java\bin\com.ibm.mq.amt.samples.jar`). These will not be removed by Add/Remove Programs.

In addition, if you specified a directory other than the default during installation, you must remove this directory from the PATH environment variable.

Setting the runtime environment

By default, the location of the AMI runtime binary files matches that of MQSeries (for example `C:\MQSeries\bin`). If you specified a different directory for the runtime files, you **must** add it to the PATH environment variable. (See also "Removing the AMI".)

To use the samples, add the sample C and C++ binary directories to your PATH environment variable. For example (assuming that the root directory for MQSeries is `C:\MQSeries`):

```
set PATH=%PATH%;C:\MQSeries\amt\samples\C\bin;
C:\MQSeries\amt\samples\Cpp\bin;
```

When running Java, the AMI classes (`C:\MQSeries\java\lib\com.ibm.mq.amt.jar`) and samples (`C:\MQSeries\amt\samples\java\bin\com.ibm.mq.amt.samples.jar`) must be contained in the CLASSPATH environment variable. This is done by the **setup** program.

Next step

Now go to "Local host and repository files (AS/400, UNIX, and Windows)" on page 450 to continue the installation procedure.

Directory structure (Windows)

On Windows platforms the directory structure contains:

```
\amt\amtsdfts.tst : MQSeries mqsc command file to create default MQSeries
  objects required by the AMI

\amt\amthost.xml : Sample AMI XML file used as the default host file

\amt\amt.dtd : AMI Document Type Definition file on which the AMI
  repository is based

\amt\include
  amtc.h : The C header file for the AMI
  amtcpp.hpp : The C++ header file for the AMI
  oamasami.h: The C header file for the OAMAS AMI subset

\amt\ipla : The International Program License Agreement file
\amt\li : The License Information file

\java\lib
  com.ibm.mq.amt.jar : The jar file containing the AMI classes for Java

\bin
  amt.dll : The main AMI library
  amt.lib : The AMI LIB file used for building C programs
  amtXML310.dll : The AMI XML parsing library
  amtCpp.dll : The AMI C++ library
  amtCpp.lib : The AMI LIB file used for building C++ programs
  amtJava.dll: The AMI JNI library
  amtICUUC140.dll : The AMI codepage translation library
  amtICUDATA.dll: The AMI codepage translation data library

  MSVCRT.DLL : Main MVSC runtime library
  MSVCIRT.DLL : Iostream MSVC runtime library

\amt\samples
  amtsamp.tst : MQSeries mqsc command file to create MQSeries objects
  required by AMI samples
  amt.xml : Sample AMI XML repository for use with the AMI samples

\amt\samples\C
  amtsosnd.c : C source for object-level send and forget sample
  amtsorcvc.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
  amtshrcvc.c : C source for high-level receiver sample
  amtshclt.c : C source for high-level client sample
  amtshsvr.c : C source for high-level server sample
  amtshpub.c : C source for high-level publisher sample
  amtshsub.c : C source for high-level subscriber sample
```

Installation on Windows

```
amtshfsn.c : C source for high-level send file sample
amtshfrc.c : C source for high-level receive file sample

\amt\samples\C\bin
  amtsosnd.exe : C object-level send and forget sample program
  amtsorcv.exe : C object-level receiver sample program
  amtsoclt.exe : C object-level client sample program
  amtsosvr.exe : C object-level server sample program
  amtsopub.exe : C object-level publisher sample program
  amtsosub.exe : C object-level subscriber sample program
  amtsosfsn.exe : C object-level send file sample program
  amtsosfrc.exe : C object-level receive file sample program
  amtsosgs.exe : C object-level send group sample program
  amtsosgr.exe : C object-level receive group sample program
  amtshsnd.exe : C high-level send and forget sample program
  amtshrcv.exe : C high-level receiver sample program
  amtshclt.exe : C high-level client sample program
  amtshsvr.exe : C high-level server sample program
  amtshpub.exe : C high-level publisher sample program
  amtshsub.exe : C high-level subscriber sample program
  amtshfsn.exe : C high-level send file sample program
  amtshfrc.exe : C high-level receive file sample program

\amt\samples\Cpp
  SendAndForget.cpp : C++ source for send and forget sample
  Receiver.cpp : C++ source for receiver sample
  Client.cpp : C++ source for client sample
  Server.cpp : C++ source for server sample
  Publisher.cpp : C++ source for publisher sample
  Subscriber.cpp : C++ source for subscriber sample
  ReceiveFile.cpp : C++ source for receive file sample
  SendFile.cpp : C++ source for send file sample

\amt\samples\Cpp\bin
  SendAndForget.exe : C++ send and forget sample program
  Receiver.exe : C++ receiver sample program
  Client.exe : C++ client sample program
  Server.exe : C++ server sample program
  Publisher.exe : C++ publisher sample program
  Subscriber.exe : C++ subscriber sample program
  ReceiveFile.exe : C++ receive file sample program
  SendFile.exe : C++ send file sample program

\amt\samples\java
  SendAndForget.java : Java source for send and forget sample
  Receiver.java : Java source for receiver sample
  Client.java : Java source for client sample
  Server.java : Java source for server sample
  Publisher.java : Java source for publisher sample
  Subscriber.java : Java source for subscriber sample
  ReceiveFile.java : Java source for receive file sample
  SendFile.java : Java source for send file sample

\amt\samples\java\bin
  com.ibm.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)

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') (AS/400)
```

```
ADDENVVAR ENVVAR(AMT_HOST) VALUE('myHostFile.xml')
```

```
export AMT_REPOSITORY = myData.xml (UNIX)
```

```
export AMT_HOST = myHostFile.xml
```

```
set AMT_REPOSITORY = myData.xml (Windows)
```

```
set AMT_HOST = myHostFile.xml
```

Local host and repository files (AS/400, UNIX, and Windows)

The directories `intlFiles` and `locales`, and the `.txt` and `.cnv` files in the `locales` directory, must be located relative to the directory containing the local host file. This applies whether you are using the default directory or have overridden it as described previously.

In C++ and Java, there is an extra level of flexibility in setting the location and names of the repository and local host files. You can specify the directory in which they are located by means of a name in the constructor of the `AmSessionFactory` class:

```
AmSessionFactory(name);
```

This name is equivalent to the `AMT_DATA_PATH` environment variable. If set, the name of the `AmSessionFactory` takes precedence over the `AMT_DATA_PATH` environment variable.

The repository and local host file names can be set using methods of the `AmSessionFactory` class:

```
setRepository(name);  
setLocalHost(name);
```

These `AmSessionFactory` methods take precedence over the `AMT_REPOSITORY` and `AMT_HOST` environment variables.

Once an `AmSession` has been created using an `AmSessionFactory`, the repository and local host file names and location are set for the complete life of that `AmSession`.

Local host file

An AMI installation must have a local host file. It defines the mapping from a connection name (default or repository defined) to the name of the `MQSeries` queue manager that you want to connect to on your local machine.

If you are not using a repository, or are opening (or initializing) a session using a policy that does not define a connection, the connection name is assumed to be `defaultConnection`. Using the sample `amthost.xml` file, as shown below, this maps to an empty string that defines a connection with the default queue manager.

```
<?xml version="1.0" encoding="UTF-8"?>  
<queueManagerNames  
    defaultConnection = ""  
    connectionName1   = "queueManagerName1"  
    connectionName2   = "queueManagerName2"  
</>
```

To change the default connection to a named queue manager of your choice, such as `'QMNAME'`, edit the local host file to contain the following string:

```
defaultConnection = "QMNAME"
```

If you want a repository defined connection name, such as `connectionName1`, to provide a connection to queue manager `'QMNAME1'`, edit the local host file to contain the following string:

```
connectionName1   = "QMNAME1"
```

The repository connection names are not limited to the values shown (`connectionName1` and `connectionName2`). Any name can be used provided it is unique in both the repository and local host files, and consistent between the two.

Local host and repository files (AS/400, UNIX, and Windows)

Repository file

You can operate an AMI installation with or without a repository file. If you are using a repository file, such as the sample `amt.xml` file, you must have a corresponding `amt.dtd` file in the same directory (the local host file must be in this directory as well).

The repository file provides definitions for policies and services. If you do not use a repository file, AMI uses its built-in definitions. For more information, see “Chapter 18. Defining services and policies” on page 469.

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.

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

Local host and repository files (OS/390)

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 a UTF-8 text file best suited to editing on a workstation. If you prefer to maintain your host file on the host, you should use the AMTHOST2 sample, which is in an EBCDIC codepage.

```
<?xml version="1.0" encoding="UTF-8"?>
<queueManagerNames
  defaultConnection = ""
  connectionName1  = "queueManagerName1"
  connectionName2  = "queueManagerName2"
/>
```

To change the default connection to a named queue manager of your choice, such as 'QMNAME', edit the local host file to contain the following string:

```
defaultConnection = "QMNAME"
```

If you want a repository defined connection name, such as connectionName1, to provide a connection to queue manager 'QMNAME1', edit the local host file to contain the following string:

```
connectionName1 = "QMNAME1"
```

The repository connection names are not limited to the values shown (connectionName1 and connectionName2). Any name can be used provided it is unique in both the repository and local host files, and consistent between the two.

"Repository and local host caches" explains how to use a local host cache instead of a local host file.

Repository file

You can operate an AMI installation with or without a repository file. The repository file provides definitions for policies and services. If you do not use a repository file, AMI uses its built-in definitions. For more information, see "Chapter 18. Defining services and policies" on page 469.

"Repository and local host caches" explains how to use a repository cache instead of a repository file.

Repository and local host caches

On OS/390, you can generate caches for use instead of repository and local host files. This gives a higher performance alternative to the files, but requires some additional configuration.

Generating caches

The AMI on OS/390 includes a program (AMTASM10) that generates assembler source code defining repository and local host caches. This program runs in a similar manner to any AMI batch program, and outputs a repository cache definition to the DD name ASMREPOS, and a local host cache to the DD name ASMHOST. The cache generator issues messages to the SYSPRINT data set, and returns zero if it is successful.

Local host and repository files (OS/390)

Here is a sample JCL fragment to run the cache generator (with US English messages):

```
//GO EXEC PGM=AMTASM10
//STEPLIB DD DSN=h1q.SCSQLOAD,DISP=SHR
//        DD DSN=h1q.SCSQANLE,DISP=SHR
//AMTHOST DD DSN=h1q.SCSQPROC(AMTHOST),DISP=SHR
//AMT     DD DSN=h1q.SCSQPROC(AMT),DISP=SHR
//SYSPRINT DD SYSOUT=*
//ASMHOST DD DSN=target(AMTHOST),DISP=SHR
//ASMREPOS DD DSN=target(AMT),DISP=SHR
```

When you have generated assembler source code successfully for your repository and host file cache, you must assemble and link edit them. Messages returned by the cache generator are described in the following section.

Using a cache

When your application creates an AMI session, the AMI first tries to load caches, before it tries to open files. The module that the AMI loads has the same name as the corresponding filename, that is AMT for the repository file and AMTHOST for the local host file. You can modify the name that will be loaded using environment variables as discussed in “Batch, RRS-batch, IMS” on page 453 and “CICS” on page 453.

Batch, RRS-batch, and IMS applications must include the dataset that contains your cache in the JCL STEPLIB. There is no need to use DD AMT or DD AMTHOST statements to locate the cached files.

CICS applications must add the dataset that contains the cache to the region DFHRPL, and define the cache to CICS using the CICS supplied CEDA transaction. There is no need to define the AMT file to CICS.

Cache generator messages

The following messages are issued by the cache generator. Terms like “%li” will be printed as decimal numbers; they hold the AMI completion and reason codes.

```
"AMT0001W AMI MESSAGE MODULE NOT FOUND"
```

```
/* Explanation: */
/* The AMI failed to load its message module. */
/* User Response: */
/* Batch, IMS: Ensure that one of the language-specific datasets is */
/* in your STEPLIB concatenation. */
/* CICS: Ensure that one of the language-specific datasets is */
/* in your DFHRPL concatenation, and the message module */
/* AMTMSG10 is defined to CICS. */
/* Explanation: */
```

```
"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. */
/* Explanation: */
```

```
"AMT0003I AMI repository cache warning, AMCC=%li, AMRC=%li"
```

```
/* Explanation: */
/* An AMI operation generated a warning. */
/* User Response: */
```


Local host and repository files (OS/390)

```
/* See the MQSeries Application Messaging Interface Manual for an */
/* explanation of CompCode, AMCC, and Reason, AMRC. */
/*****/
"AMT0004I AMI repository cache created"

/*****/
/* Explanation: */
/* A repository cache was successfully created. */
/* User Response: */
/* None. */
/*****/
"AMT0005I AMI host file cache created"

/*****/
/* Explanation: */
/* A host file cache was successfully created. */
/* User Response: */
/* None. */
/*****/
```

The administration tool

The AMI administration tool is for use on Windows NT Version 4 only.

Installation

The administration tool is packaged with the AMI in `ma0f_nt.zip` and optionally installed with the AMI using the setup InstallShield program (see “Installation on Windows” on page 447). It is installed in sub-directory `amt\AMITool`.

To start the AMI administration program, select **IBM MQSeries AMI \ IBM MQSeries AMI Administration Tool** using the **Start Programs menu**, or double-click on the file `\amt\AMITool\amitool.bat`.

To verify that the tool has been installed correctly, click on **Open** in the **File** menu, navigate to the `\amt\AMITool` directory, and open the file `amiSample.xml`. You should see a number of services and policies in the navigation pane on the left. Select one of them by clicking on it, and you should see its attributes displayed in the pane on the right.

Operation

The administration tool enables you to create definitions for:

Service points used to create sender or receiver services

Distribution lists

must include at least one sender service

Publishers must include a sender service as the broker service

Subscribers must include sender and receiver services as the broker and receiver services

Policies contain sets of attributes: initialization, general, send, receive, publish, subscribe

The default attributes provided by the tool are as specified in “Service definitions” on page 472 and “Policy definitions” on page 475.

When you have entered the definitions you require, select **Save** in the **File** menu to save them as an XML-format repository file. It is recommended that you define all your services and policies in the same repository file.

The repository file must be copied to a location where it can be accessed by the AMI (see “Local host and repository files (AS/400, UNIX, and Windows)” on page 450). If the Application Messaging Interface is on the same system as the tool, the repository file can be copied to the AMI directory. Otherwise, the repository file must be transferred to that system using a method such as file sharing or FTP.

Note: To open an existing repository file (including the `amt.xml` file provided in the samples directory), the repository file and the `amt.dtd` file must both be in the same directory.

Further information can be found in the AMI administration tool online help.

Connecting to MQSeries

You can connect to MQSeries, the transport layer, using an MQSeries server or an MQSeries client. Using the default policy, the AMI automatically detects whether it should connect directly or as a client. If you have an installation that has both an MQSeries client and an MQSeries queue manager, and you want the AMI to use the client for its connection, you must specify the Connection Type as Client in the policy initialization attributes (see “Policy definitions” on page 475).

Using MQSeries Integrator Version 1

If you are using the AMI with MQSeries Integrator Version 1, the Service Type for the sender service point must be defined in the repository as MQSeries Integrator V1 (see “Service definitions” on page 472). This causes an MQRFH header containing application group and message type name/value elements to be added to a message when it is sent.

The Application Group definition is included in the policy send attributes (see “Policy definitions” on page 475). The message type is defined as the message format value set in the message object (using **amMsgSetFormat**, for example). If this is set to AMFMT_NONE, the message type is defined as the Default Format for the sender service point (a maximum of eight characters in MQSeries). If you wish to specify the message type directly, you must do this explicitly using the **amMsgAddElement** function in C, or the equivalent **addElement** method in C++ and Java. This allows you to add a message type that differs from the message format, and is more than eight characters long.

Using MQSeries Publish/Subscribe

If you want to use the publish/subscribe functions of the AMI, you must have MQSeries Publish/Subscribe installed (see the *MQSeries Publish/Subscribe User's Guide*). The Service Type for the sender and receiver service points used by the publisher and subscriber must be defined in the repository as MQRFH (see “Service definitions” on page 472). This causes an MQRFH header containing publish/subscribe name/value elements to be added to a message when it is sent.

Using MQSeries Integrator Version 2

You can use your existing AMI repository file, MQSeries Publish/Subscribe applications, and 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.

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 recognise the broker being used. The AMI performs the required mapping of MQSeries Integrator Version 2 response values as necessary.

Migrating to MQSeries Integrator V2 from V1 and MQSeries Publish/Subscribe

MQSeries Integrator V2 will support applications written to use MQSI V1 and MQSeries Publish/Subscribe. Existing AMI applications and the Service Type in the repository Service Point definitions do not therefore need to be changed.

Connecting to MQSeries

Applications that want to exploit new functions in MQSI V2 should have their Service Point definitions changed to a Service Type of 'MQSeries Integrator V2' and, if necessary, use the new AMI calls and parameters.

Existing publish/subscribe applications that have used the element calls to explicitly add name value pairs to the MQRFH can continue to use the same names for the elements when migrating to MQSI V2.

Creating default MQSeries objects

The Application Messaging Interface makes use of default MQSeries objects, which must be created before using the AMI. To do this, you run the MQSC script `amtsdfts.tst` (you might want to edit this file first, to suit the requirements of your installation).

For AS/400, start the the local queue manager by typing the following on the command line, where `QMName` is the name of your MQSeries queue manager:

```
STRMQM MQMNAME(QMName)
```

Then run the default MQSC script by typing the following command:

```
STRMQMQSC 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:

```
//COMMAND EXEC PGM=CSQUTIL,PARM='QMGR'  
//STEPLIB DD DSN=h1q.SCSQAUTH,DISP=SHR  
// DD DSN=h1q.SQSCANLE,DISP=SHR  
//AMTSDFTS DD DSN=h1q.SCSQPROC(AMTSDFTS),DISP=SHR  
//SYSPRINT DD SYSOUT=*  
//SYSIN DD *  
COMMAND DDNAME(AMTSDFTS)  
/*
```

where `h1q` 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)
```

where `{QMName}` is the name of your MQSeries queue manager and `{Location}` is the location of the `amtsdfts.tst` file.

The sample programs

Sample programs are provided to illustrate the use of the Application Messaging Interface.

It is recommended that you run one or more of the sample programs to verify that you have installed the Application Messaging Interface correctly.

If you are using the OS/390 platform, go to “Sample programs for OS/390” on page 464.

Sample programs for AS/400, UNIX, and Windows

There are ten basic sample programs for AS/400, UNIX, and Windows platforms, performing approximately the same function in C, C++, and Java. Consult the source code to find out how the programs achieve this functionality. The C samples are provided for both the high-level interface and the object interface.

Table 5. The sample programs for AS/400, UNIX, and Windows platforms

Description	C high-level	C Object-level	C++	Java
A sample that sends a datagram message, expecting no reply.	amtshsnd	amtsosnd	SendAndForget (for AS/400, SendForget)	SendAndForget
A sample that receives a message, with no selection.	amtshrcv	amtsorcvc	Receiver	Receiver
A sample that sends a request and receives a reply to this request (a simple client program).	amtshclt	amtsoclt	Client	Client
A sample that receives requests and sends replies to these requests (a simple server program).	amtshsvr	amtsosvr	Server	Server
A sample that periodically publishes information on the weather.	amtshpub	amtsopub	Publisher	Publisher
A sample that subscribes to information on the weather, and receives publications based on this subscription.	amtshsub	amtsosub	Subscriber	Subscriber
A sample that sends messages using simulated group support.	-	amtsosgs	-	-
A sample that receives messages using simulated group support.	-	amtsosgr	-	-
A sample that performs a file transfer send on a user supplied text file.	amtshfsn	amtsofsn	SendFile	SendFile
A sample that performs a file transfer receive on a user supplied text file.	amtshfrc	amtsofrc	ReceiveFile (for AS/400, RcvFile)	ReceiveFile

To find the source code and the executables for the samples, see “Directory structure” on page 430 (AIX), page 433 (AS/400), page 438 (HP-UX), page 445 (Solaris), and page 448 (Windows).

The sample programs

Running the AS/400, UNIX, and Windows sample programs

Before you can run the sample programs on AS/400, UNIX, or Windows platforms, you must make a number of preparations. These are described in the following sections.

MQSeries objects

The sample programs require some MQSeries objects to be defined. To do this, use an MQSeries MQSC file, `amtsamp.tst`, which is shipped with the samples.

For AS/400, start the the local queue manager by typing the following on the command line:

```
STRMQM QMNAME(QMName)
```

where `QMName` is the name of your MQSeries queue manager.

Then run the default MQSC script by typing the following command:

```
STRMQMMQSC SRCMBR(AMTSAMP) SRCFILE(QMQMAMI/AMTMQSC) QMNAME(QMName)
```

For UNIX or Windows, start the local queue manager by typing the following at a command line:

```
strmqm {QMName}
```

where `{QMName}` is the name of your MQSeries queue manager.

Then run the sample MQSC script by typing one of the following:

```
runmqsc {QMName} < {Location}/amtsamp.tst (UNIX)
```

```
runmqsc {QMName} < {Location}\amtsamp.tst (Windows)
```

where `{QMName}` is the name of your MQSeries queue manager and `{Location}` is the location of the `amtsamp.tst` file.

Repository and host files

Copy the sample repository file, `amt.xml`, into the default location for your platform (see “Local host and repository files (AS/400, UNIX, and Windows)” on page 450).

Modify the host file so that your MQSeries queue manager name, `{QMName}`, is known as `defaultConnection`.

Running the publish/subscribe samples

To run the AMI publish/subscribe samples, you need access to an MQSeries broker. This can be either MQSeries Publish/Subscribe or MQSeries Integrator Version 2. You can issue publish/subscribe requests locally or remotely to either broker. If the platform on which the requesting application runs does not support an MQSeries broker, publish/subscribe requests can only be issued remotely. In this situation, you must set up the appropriate MQSeries channels, and ensure that the remote queue manager and channels are started.

MQSeries Publish/Subscribe broker: To run the publish/subscribe samples with MQSeries Publish/Subscribe broker, you must start the broker. Type the following at a command line:

```
strmqbrk -m {QMName}
```

where `{QMName}` is the name of your MQSeries queue manager.

MQSeries Integrator Version 2: To run the publish/subscribe samples with MQSeries Integrator Version 2 you need to do the following:

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}
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.
4. Add the new message flow to the execution group for your Broker and deploy it.

For further details, refer to the *MQSeries Integrator Version 2.0 Programming Guide*.

Setting the runtime environment

Before you run the AMI samples, make sure that you have set up the runtime environment. See “Setting the runtime environment” on page 429 (AIX), page (AS/400), page 437 (HP-UX), page 444 (Solaris), and page 447 (Windows).

Running the C and C++ samples

You can run a C or C++ sample program by typing the name of its executable at a command line. For example:

```
amtsosnd
```

will run the “Send and forget” sample written using the C object interface.

Running the Java samples

The AMI samples for Java are in a package called:

```
com.ibm.mq.amt.samples
```

To invoke them, you need to specify the name of the sample plus its package name. For example, to run the “Send and forget” sample, use:

```
java com.ibm.mq.amt.samples.SendAndForget
```

Running the sample programs (AS/400)

Executable sample programs are provided in the QMQMAMI library. To run C and C++ samples on the AS/400, use CALL, followed by the name of the executable. For example, to run the “Send and Forget” sample written using the C object interface, enter:

```
CALL AMTSOSND
```


The sample programs

Sample programs for OS/390

There are ten basic sample programs in C for the OS/390 platform, and a matching set in COBOL that perform approximately the same function. Consult the source code to find out how the programs achieve this functionality. The samples are provided for both the high-level interface and the object-level interface in most cases.

There is also a C header file `amts39sp` that implements environment-specific I/O functions for CICS and IMS. This header file is not required to build the samples for Batch.

Table 6. The sample programs for OS/390 ('batch' includes RRS-batch)

Description	C High level	C Object level	COBOL High level	COBOL Object level
A sample that sends a datagram message, expecting no reply.	AMTSHSND	AMTSOSND	AMTVHSND	AMTVOSND
A sample that receives a message, with no selection.	AMTSHRCV	AMTSORCV	AMTVHRCV	AMTVORCV
A sample that sends a request and receives a reply to this request (a simple client program).	AMTSHCLT	AMTSOCLT	AMTVHCLT	AMTVOCLT
A sample that receives requests and sends replies to these requests (a simple server program).	AMTSHSVR	AMTSOSVR	AMTVHSVR	AMTVOSVR
A sample that periodically publishes information on the weather.	AMTSH PUB	AMTSOPUB	AMTVHPUB	AMTVOPUB
A sample that subscribes to information on the weather, and receives publications based on this subscription.	AMTSHSUB	AMTSOSUB	AMTVH SUB	AMTVOSUB
A sample that sends simulated group messages. This uses object-level calls only.	Not applicable	AMTSOSGS	Not applicable	AMTVOSGS
A sample that receives simulated group messages. This uses object-level calls only.	Not applicable	AMTSOSGR	Not applicable	AMTVOSGR
A sample that performs a file transfer send on a user-supplied text file. Not for use under CICS.	AMTSHFSN	AMTSOFSN	AMTVHFSN	AMTVOFSN
A sample that performs a file transfer receive on a user-supplied text file. Not for use under CICS.	AMTSHFRC	AMTSOFRC	AMTVHFRC	AMTVOFRC

To find the source code for the samples, see "Directory structure (OS/390)" on page 441.

Running the sample programs (OS/390)

Before you can run the sample programs on the OS/390 platform, there are a number of actions to take.

Building the sample programs

The samples for OS/390 are provided as source code only, so you must build them before you can run them. See “Building C applications” on page 29 and “COBOL applications on OS/390” on page 241.

MQSeries objects

The sample programs require some MQSeries objects to be defined. This can be done with an MQSeries MQSC file, AMTSAMP, which is shipped with the samples.

First start the local queue manager, as described in the *MQSeries for OS/390 System Administration Guide*. If you are using the CICS environment, ensure that the MQSeries CICS adapter is set up and the CICS region is connected to the queue manager.

Then run the sample MQSC script AMTSAMP (located in the h1q.SCSQPROC dataset) using the MQSeries utility program CSQUTIL. Following is a JCL fragment to help you run the utility:

```
//COMMAND EXEC PGM=CSQUTIL,PARM='QMGR'
//STEPLIB DD DSN=h1q.SCSQAUTH,DISP=SHR
// DD DSN=h1q.SQSCANLE,DISP=SHR
//AMTSAMP DD DSN=h1q.SCSQPROC(AMTSAMP),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COMMAND DDNAME(AMTSAMP)
/*
```

where h1q is the high level qualifier of your MQSeries installation, and QMGR is your queue manager name.

Repository and host files

The sample repository AMT (located in h1q.SCSQPROC) is appropriate for use with all the sample programs (though many of the samples will work correctly without a repository). If you wish to use the repository file, ensure that the sample program has access to it, as described in “Local host and repository files (OS/390)” on page 453.

For batch, RRS-batch, and IMS programs (not CICS), copy the sample host file AMTHOST (UTF-8) or AMTHOST (EBCDIC) from h1q.SCSQPROC to another location, and modify it so that your MQSeries queue manager name is defaultConnection. Ensure that the sample program has access to the host file, using DD statements as described in “Local host and repository files (OS/390)” on page 453.

Running the publish/subscribe samples

To use the publish/subscribe samples, you need access to an MQSeries Publish/Subscribe broker. Because this is not available on OS/390, you must have an MQSeries queue manager and publish/subscribe broker running on another platform. In this situation, you must set up the appropriate MQSeries channels, and ensure that the remote queue manager and channels are started.

MQSeries Publish/Subscribe broker: To run the publish/subscribe samples with MQSeries Publish/Subscribe broker, you must start the broker. Type the following at a command line:

```
strmqbrk -m {QMName}
```

The sample programs

where {QMName} is the name of your MQSeries queue manager.

MQSeries Integrator Version 2: To run the publish/subscribe samples with MQSeries Integrator Version 2 you need to do the following:

1. Start the broker and the Configuration Manager. Type the following at a command line, where {BrokerName} is 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
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 440.

File name input for the file transfer samples

There are 3 ways in MVS to specify the file name for the file transfer samples:

1. Use single quotes.

```
// PARM='MYTEST.FILE'
```

It will then obey MVS rules and be prefixed with the user's RACF[®] ID as the high level qualifier.

2. Supply a fully qualified filename using double quotes, with an extra outer pair to contain the parm data.

```
// PARM= "userId.MYTEST.FILE"
```

3. Supply the keywords DD:FILE in the parms where FILE is the DD NAME.

```
// PARM='DD:MYFILE
.
.
// MYFILE DD DSNAME=userId.MYTEST.FILE,DISP=SHR
```

Each method resolves to userId.MYTEST.FILE.

Running the batch samples

You can run batch sample programs by constructing a piece of JCL to run the program, and submitting that JCL from ISPF. The batch samples can also be used as RRS-batch sample programs.

Running the CICS samples

Ensure that the CICS DFHRPL includes the load library containing the sample, as well as the AMI library. Define the sample program to CICS, as well as a transaction to run the program. Finally, ensure that the AMI library, sample program and sample transaction are installed in your CICS region. Type the transaction name into a CICS console to run the sample.

Running the IMS samples

Ensure that the IMS message processing region JCL includes the load library that contains the sample, as well as the AMI library. Define the sample program and transaction name to IMS. Type the transaction name into an IMS console to run the sample.

Chapter 18. Defining services and policies

Definitions of services and policies created by a system administrator are held in a *repository*. The Application Messaging Interface provides a tool to enable the administrator to set up new services and policies, and to specify their attributes (see “The administration tool” on page 457).

This chapter contains:

- “Services and policies”
- “Service definitions” on page 472
- “Policy definitions” on page 475

Services and policies

A repository file contains definitions for *policies* and *services*. A service is the generic name for any object to which a send or receive request can be issued, that is:

- Sender
- Receiver
- Distribution list
- Subscriber
- Publisher

Sender and receiver definitions are represented in the repository by a single definition called a *service point*.

Policies, and services other than distribution lists, can be created with or without a corresponding repository definition; distribution lists can be created only with a corresponding repository definition.

To create a service or policy using the repository, the repository must contain a definition of the appropriate type with a name that matches the name specified by the application. To create a sender object named ‘DEBITS’ (using **amSesCreateSender** in C, for example) the repository must have a service point definition named ‘DEBITS’.

Policies and services created with a repository have their contents initialized from the named repository definition.

If the repository does not contain a matching name, a warning is issued (such as AMRC_POLICY_NOT_IN_REPOS). The service or policy is then created without using the repository (unless it is a distribution list).

Policies and services created without a repository (either for the previous reason, or because the repository is not used), have their contents initialized from one of the system provided definitions (see “System provided definitions” on page 470).

Definition names in the repository must not start with the characters ‘AMT’ or ‘SYSTEM’.

Services and policies

System provided definitions

The AMI provides a set of definitions for creating services and policies without reference to a repository.

Table 7. System provided definitions

Definition	Description
AMT.SYSTEM.POLICY	This provides a policy definition with the defaults specified in "Policy definitions" on page 475, except that Wait Interval Read Only is not selected in the Receive attributes.
AMT.SYSTEM.SYNCPOINT.POLICY	This provides a policy definition the same as AMT.SYSTEM.POLICY, except that Syncpoint is selected in the General attributes.
AMT.SYSTEM.SENDER	This provides a sender definition with the defaults specified in "Service definitions" on page 472, with the Queue Name the same as the Sender object.
AMT.SYSTEM.RESPONSE.SENDER	This provides a sender definition the same as AMT.SYSTEM.SENDER, except that Definition Type, Queue Name and Queue Manager Name are set to 'Undefined' (that is, set when used).
AMT.SYSTEM.RECEIVER	This provides a receiver definition the same as AMT.SYSTEM.SENDER.
AMT.SYSTEM.SUBSCRIBER	This provides a subscriber definition in which the Sender Service has the same name as the Subscriber object, and the Receiver Service has the same name with the suffix '.RECEIVER'.
AMT.SYSTEM.PUBLISHER	This provides a publisher definition in which the Broker Service has the same name as the Publisher object.

System default objects

A set of system default objects is created at session creation time. This removes the overhead of creating the objects from applications using these defaults. The system default objects are available for use from the high-level and object-level interfaces in C. They cannot be accessed using C++ or Java (these languages can use the built-in definitions to create an equivalent set of objects if required).

The default objects are created using the system provided definitions, as shown in the following table.

Table 8. System default objects

Default object	Definition
SYSTEM.DEFAULT.POLICY	AMT.SYSTEM.POLICY
SYSTEM.DEFAULT.SYNCPOINT.POLICY	AMT.SYSTEM.SYNCPOINT.POLICY
SYSTEM.DEFAULT.SENDER	AMT.SYSTEM.SENDER
SYSTEM.DEFAULT.RESPONSE.SENDER	AMT.SYSTEM.RESPONSE.SENDER
SYSTEM.DEFAULT.RECEIVER	AMT.SYSTEM.RECEIVER
SYSTEM.DEFAULT.SUBSCRIBER	AMT.SYSTEM.SUBSCRIBER
SYSTEM.DEFAULT.PUBLISHER	AMT.SYSTEM.PUBLISHER

Table 8. System default objects (continued)

Default object	Definition
SYSTEM.DEFAULT.SEND.MESSAGE	N/A
SYSTEM.DEFAULT.RECEIVE.MESSAGE	N/A

The default objects can be used explicitly using the AMI constants (see “Appendix B. Constants” on page 509), or used to provide defaults if a particular parameter is omitted (by specifying NULL, for instance).

Handle synonyms are also provided for these objects, for use from the object interface (see “Appendix B. Constants” on page 509). Note that the first parameter on a call must be a real handle; you cannot use a synonym handle in this case.

Service definitions

This section gives the service definitions for:

- service point (sender/receiver)
- distribution list
- subscriber
- publisher

Service point (sender/receiver)

Table 9. Service point (sender/receiver)

Attribute	Comments
Name	Mandatory name, specified on AMI calls. 1
Queue Name	Mandatory name of the queue representing the service that messages are sent to or received from. 2
Queue Manager Name	Name of the queue manager that owns Queue Name. If blank, the local queue manager name is used. 2
Model Queue Name	Name of a model queue definition used to create a dynamic queue (normally a Reply Service to receive response messages). Required if the Definition Type is 'Dynamic'. 2
Dynamic Queue Prefix	Name of a prefix used when creating a dynamic queue from Model Queue Name. Required if the Definition Type is 'Dynamic'. If the last non-blank character in positions 1 to 33 of the prefix is '*', the '*' is replaced by a string that guarantees that the name generated is unique. 2
Definition Type	Defines how the AMI obtains the queue name for the service point. If set to 'Predefined' (the default), the Queue Name and Queue Manager Name as specified earlier are used. If set to 'Dynamic', the Model Queue Name and Dynamic Queue Prefix are used to create a dynamic queue.
Service Type	<p>Defines the RF header (if any) that is sent with the message data, and the parameters within the header.</p> <p>Set to 'Native' for a native MQ service (default).</p> <p>Set to 'MQSeries Integrator V1' for MQSeries Integrator Version 1 (adds the OPT_APP_GROUP and OPT_MSG_TYPE fields to the MQRFH header).</p> <p>Set to 'RF Header V1' for MQSeries Publish/Subscribe applications.</p> <p>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.</p> <p>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.</p>
Default Format	Optional format name to insert in the MQMD, if a format value of FMT_NONE is set in the message object. Also used as the MsgType when the service is an MQSeries Integrator Version 1 broker, if AMFMT_NONE is set in the message object and the MsgType has not been added explicitly (using amMsgAddElement or equivalent). 3
Default MCD Domain	Defines the default message service domain value. This is added to any message being sent using this service point if the Service Type is 'MQSeries Integrator V2', the value of this field is non-blank and a message service domain element has not been explicitly added to the message by the application. 1 4

Table 9. Service point (sender/receiver) (continued)

Attribute	Comments
Default MCD Set	Defines the default message set value. This is added to any message being sent using this service point if the Service Type is '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 4
Default MCD Type	Defines the default message type value. This is added to any message being sent using this service point if the Service Type is '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.)
<p>Notes:</p> <p>1 The name is a maximum of 256 characters, and can contain the following characters: A-Z, a-z, 0-9, '.', '/', '_', and '%'.</p> <p>2 The name is a maximum of 48 characters, and can contain the following characters: A-Z, a-z, 0-9, '.', '/', '_', and '%'.</p> <p>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).</p> <p>4 This attribute is applicable only for Service Type 'MQSeries Integrator V2' and is ignored for other Service Type settings.</p> <p>5 The name is a maximum of 6 characters, and can contain any numeric character.</p>	

Service definitions

Distribution list

Table 10. Distribution list

Attribute	Comments
Name	Mandatory name, specified on AMI calls. 1
Available Service Points	List of service points that make up the distribution list. They must be valid service point names.
Note: 1 The name is a maximum of 256 characters, and can contain the following characters: A-Z, a-z, 0-9, '.', '/', '_', and '%'.	

Subscriber

Table 11. Subscriber

Attribute	Comments
Name	Mandatory name, specified on AMI calls. 1
Sender Service	The name of the sender service that defines the publish/subscribe broker. It must be a valid service point name.
Receiver Service	The name of the receiver service that defines where publication messages are to be sent. It must be a valid service point name.
Note: 1 The name is a maximum of 256 characters, and can contain the following characters: A-Z, a-z, 0-9, '.', '/', '_', and '%'.	

Publisher

Table 12. Publisher

Attribute	Comments
Name	Mandatory name, specified on AMI calls. 1
Sender Service	The name of a sender service that defines the publish/subscribe broker. It must be a valid service point name.
Note: 1 The name is a maximum of 256 characters, and can contain the following characters: A-Z, a-z, 0-9, '.', '/', '_', and '%'.	

Policy definitions

This section describes the policy definitions for the following attributes:

- initialization
- general
- send
- receive
- subscribe
- publish

Initialization attributes

Table 13. Initialization attributes

Attribute	Comments
Name	Mandatory policy name, specified on AMI calls. 1
Connection Name	If Connection Mode is set to 'Real', Connection Name is the name of the queue manager the application will connect to. If blank, the default local queue manager is used. If Connection Mode is 'Logical', the Connection Name attribute is required and is the name of the logical connection used with the local host file to generate the queue manager to which connection is made. 2
Connection Mode	If Connection Mode is set to 'Real' (the default), Connection Name is used as the queue manager name for connection. If Connection Mode is set to 'Logical', Connection Name is used as a key to the host file on the system where the application is running that maps Connection Name to a queue manager name. This allows applications running on different systems in the network to use the same repository (connection name) to connect to different local queue managers.
Connection Type	If Connection Type is set to 'Auto' (the default), the application automatically detects if it should connect directly, or as a client. If Connection Type is 'Client', the application connects as a client. If Connection Type is 'Server', the application connects directly to the queue manager.
Trusted Option	If set to 'Normal' (the default), no fastpath is used. If set to 'Trusted', the application can use fastpath facilities that might compromise integrity. This option is only supported on Windows.
Notes:	
1 The name is a maximum of 256 characters, and can contain the following characters: A-Z, a-z, 0-9, '.', '/', '_', and '%'.	
2 The name is a maximum of 48 characters, and can contain the following characters: A-Z, a-z, 0-9, '.', '/', '_', and '%'.	

Policy definitions

General attributes

Table 14. General attributes

Attribute	Comments
Message Context	<p>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).</p> <p>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.</p>
Syncpoint	If selected, the send or receive is part of a unit of work (default is 'not selected').

Send attributes

Table 15. Send attributes

Attribute	Values	Default	Comments
Implicit Open	Selected Not selected	Selected	When selected, the queue is opened implicitly (must be selected for the C and COBOL high-level interfaces). 1
Leave Queue Open	Selected Not selected	Selected	When selected, a queue that was implicitly opened will be left open. 1
Priority	0-9 As Transport	As Transport	<p>The priority set in the message, where 0 is the lowest priority and 9 is the highest.</p> <p>When set to As Transport, the value from the queue definition is used.</p> <p>You must deselect As Transport before you can set a priority value.</p>
Persistence	Yes No As Transport	As Transport	<p>The persistence set in the message, where Yes is persistent and No is not persistent.</p> <p>When set to As Transport, the value from the underlying queue definition is used.</p>
Expiry Interval	0-999999999 Unlimited	Unlimited	A period of time (in tenths of a second) after which the message will not be delivered.
Retry Count	0-999999999	0	The number of times a send will be retried if the return code gives a temporary error. Retry is attempted under the following conditions: Queue full, Queue disabled for put, Queue in use.
Retry Interval	0-999999999	1000	The interval (in milliseconds) between each retry.
Response Correl Id	Message Id Correl Id	Message Id	The Id set in the Correl Id of a response or report message. This is set to either the Message Id or the Correl Id of the request message, as specified.
Exception Action	Discard DLQ	DLQ	Action when a message cannot be delivered. When set to DLQ, the message is sent to the dead-letter queue. When set to Discard, it is discarded.
Report Data	Report With Data With Full Data	Report	The amount of data included in a report message, where Report specifies no data, With Data specifies the first 100 bytes, and With Full Data specifies all data.
Report Type Exception	Selected Not selected	Not selected	When selected, Exception reports are required.

Table 15. Send attributes (continued)

Attribute	Values	Default	Comments
Report Type COA	Selected Not selected	Not selected	When selected, Confirm on Arrival reports are required.
Report Type COD	Selected Not selected	Not selected	When selected, Confirm on Delivery reports are required.
Report Type Expiry	Selected Not selected	Not selected	When selected, Expiry reports are required.
Segmentation	Selected Not selected	Not selected	When selected, Segmentation of the message is allowed.
Split File	Logical Physical	Physical	<p>When set to Logical, the file is split into separate messages at record boundaries. On Windows, HP-UX, AIX, and Sun Solaris, this is the end of a line. On OS/390, this is a record boundary.</p> <p>When set to Physical, the file is split into separate messages at boundaries that are determined by AMI.</p>
Bind On Open	Yes No As Transport	As Transport	<p>Bind On Open controls the binding of a service point to a particular instance of an MQSeries cluster queue.</p> <p>When set to Yes, the service point is bound to the destination queue when the service is opened.</p> <p>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.</p> <p>When set to As Transport, the value from the underlying queue definition is used.</p>
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	0-999999999 Unlimited	Unlimited	A period of time (in milliseconds) that the receive waits for a message to be available.
Wait Interval Read Only	Selected Not selected	Selected	When selected, an application cannot override the Wait Interval value in the policy object.
Convert	Selected Not selected	Selected	When selected, the message is code page converted by the message transport when received.
Wait For Whole Group	Selected Not selected	Selected	When selected, all messages in a group must be available before any message is returned by the receive. When not selected, AMRC_NO_MSG_AVAILABLE may be returned to the application before the complete group is received. In this case, any simulated group state information is destroyed and any remaining messages in a simulated group are orphaned.
Handle Poison Message	Selected Not selected	Selected	When selected, poison message handling is enabled. 1
Accept Truncated Message	Selected Not selected	Not selected	When selected, truncated messages are accepted.
Open Shared	Selected Not selected	Selected	When selected, the queue is opened as a shared queue.
File Disposition	New Overwrite Append	New	Specifies whether an incoming file is created as a New file, Overwrites an existing file, or becomes an Append to an existing file.

Table 16. Receive attributes (continued)

Attribute	Values	Default	Comments
Note:			
<p>1 A poison message is one for which the count of the number of times it has been backed-out during a unit of work exceeds the maximum backout limit specified by the underlying MQSeries transport queue object. If poison message handling is enabled during a receive request the AMI will handle it as follows:</p> <p>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.</p> <p>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.</p> <p>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.</p>			

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.

Policy definitions

Publish attributes

Table 18. Publish attributes

Option	Values	Default	Comments
Retain	Selected Not selected	Not selected	When selected, the publication is retained by the broker.
Publish To Others Only	Selected Not selected	Not selected	When selected, the publication is not sent to the publisher if it has subscribed to the same topic (used for conference-type applications).
Suppress Registration	Selected Not selected	Selected	When selected, implicit registration of the publisher is suppressed. (This attribute is ignored for MQSeries Integrator Version 2.)
Publish Locally	Selected Not selected	Not selected	When selected, the publication is only sent to subscribers that are local to the broker.
Accept Direct Requests	Selected Not selected	Not selected	When selected, the publisher should accept direct requests from subscribers.
Anonymous Registration	Selected Not selected	Not selected	When selected, the publisher registers anonymously.
Use Correl Id As Id	Selected Not selected	Not selected	When selected, the Correl Id is used by the broker as part of the publisher's identity.

Chapter 19. Problem determination

This chapter shows you how to use the trace facility in the Application Messaging Interface, and gives some information about finding the causes of problems. See:

- “Using trace (AS/400, UNIX, and Windows)”
- “Using trace (OS/390)” on page 489
- “When your AMI program fails” on page 492

Using trace (AS/400, UNIX, and Windows)

The Application Messaging Interface includes a trace facility to help identify what is happening when you have a problem. It shows the paths taken when you run your AMI program. Unless you have a problem, you are recommended to run with tracing set off to avoid any unnecessary overheads on your system resources.

There are three environment variables that you set to control trace:

```
AMT_TRACE
AMT_TRACE_PATH
AMT_TRACE_LEVEL
```

For AS/400, you set these environment variables using the following commands:

```
ADDENVVAR - Adds an environment variable
CHGENVVAR - Changes an environment variable
WRKENVVAR - Displays an environment variable
RMVENVVAR - Deletes an environment variable
```

To set global environment variables, specify LEVEL(*SYS) in the ADDENVVAR command, for example:

```
ADDENVVAR ENVVAR(variable) VALUE(value) LEVEL(*SYS)
```

Alternatively, you can create a CL program that contains commands to set the environment variables. At startup, you can run this program by specifying the name of the CL program with the SYSVAL QSTRUPPGM command, for example:

```
CHGSYSVAL SYSVAL(QSTRUPPGM) VALUE('program')
```

For UNIX or Windows, you set these variables in one of two ways.

1. From a command prompt. The settings are locally effective, so you must then start your AMI program from this prompt.
If you use the export command with the AS/400 Qshell interpreter, you must specify the -s option to set the environment in the current process.
2. By putting the information into your system startup file; these settings are globally effective. To do this:
 - Select Main -> Control Panel on Windows NT and Windows 98
 - Edit your .profile file on UNIX systems

When deciding where you want the trace files written, ensure that the user has sufficient authority to write to, not just read from, the disk.

If you have tracing switched on, it will slow down the running of your AMI program, but it will not affect the performance of your MQSeries environment.

Using trace (AS/400, UNIX, and Windows)

When you no longer need a trace file, it is your responsibility to delete it. You must stop your AMI program running to change the status of the AMT_TRACE variable. The AMI trace environment variable is different to the trace environment variable used within the MQSeries range of products. Within the AMI, the trace environment variable turns tracing on. If you set the variable to a string of characters (any string of characters) tracing will remain switched on. It is not until you set the variable to NULL that tracing is turned off.

Trace filename and directory

The trace file name takes the form AMTnnnnn.trc, where nnnnn is the ID of the AMI process running at the time.

Commands on AS/400

WRKENVVAR

Displays the settings of all environment variables.

ADDENVVAR ENVVAR(AMT_TRACE_PATH) VALUE('/directory')

Sets the trace directory where the trace file will be written.

RMVENVVAR ENVVAR(AMT_TRACE_PATH)

Removes the AMT_TRACE_PATH environment variable; the trace file is written to the current working directory (when the AMI program was started).

ADDENVVAR ENVVAR(AMT_TRACE_LEVEL) VALUE(n)

Sets the trace level, where n is an integer from 0 through 9. 0 represents minimal tracing, and 9 represents a fully detailed trace.

You can also suffix the value with a + (plus) or - (minus) sign. When the plus sign is suffixed, the trace includes all control block dump information and all informational messages. When the minus sign is suffixed, the trace includes only the entry and exit points in the trace, with no control block information or text output to the trace file.

RMVENVVAR ENVVAR(AMT_TRACE_LEVEL)

Removes the AMT_TRACE_LEVEL environment variable. The trace level is set to its default value of 2.

ADDENVVAR ENVVAR(AMT_TRACE) VALUE(xxxxxxxx)

Sets tracing ON by putting one or more characters for the VALUE parameter. For example:

```
ADDENVVAR ENVVAR(AMT_TRACE) VALUE(yes)
```

```
ADDENVVAR ENVVAR(AMT_TRACE) VALUE(no)
```

In both of these examples, tracing will be set ON.

RMVENVVAR ENVVAR(AMT_TRACE)

Sets tracing off.

Commands on UNIX

export AMT_TRACE_PATH=/directory

Sets the trace directory where the trace file will be written.

unset AMT_TRACE_PATH

Removes the AMT_TRACE_PATH environment variable; the trace file is written to the current working directory (when the AMI program was started).

Using trace (AS/400, UNIX, and Windows)

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=xxxxxxx

Sets tracing ON by putting one or more characters after the '=' sign. For example:

```
export AMT_TRACE=yes
export AMT_TRACE=no
```

In both of these examples, tracing will be set ON.

unset AMT_TRACE

Sets tracing off.

echo \$AMT_TRACE

Displays the contents of the environment variable.

Commands on Windows

SET AMT_TRACE_PATH=drive:\directory

Sets the trace directory where the trace file will be written.

SET AMT_TRACE_PATH=

Removes the AMT_TRACE_PATH environment variable; the trace file is written to the current working directory (when the AMI program was started).

SET AMT_TRACE_PATH

Displays the current setting of the trace directory.

SET AMT_TRACE_LEVEL=n

Sets the trace level, where n is an integer from 0 through 9. 0 represents minimal tracing, and 9 represents a fully detailed trace.

You can also suffix the value with a + (plus) or - (minus) sign. When the plus sign is suffixed, the trace includes all control block dump information and all informational messages. When the minus sign is suffixed, the trace includes only the entry and exit points in the trace, with no control block information or text output to the trace file.

SET AMT_TRACE_LEVEL=

Removes the AMT_TRACE_LEVEL environment variable. The trace level is set to its default value of 2.

Using trace (AS/400, UNIX, and Windows)

SET AMT_TRACE_LEVEL

Displays the current setting of the trace level.

SET AMT_TRACE=xxxxxxx

Sets tracing ON by putting one or more characters after the '=' sign. For example:

SET AMT_TRACE=yes

SET AMT_TRACE=no

In both of these examples, tracing will be set ON.

SET AMT_TRACE=

Sets tracing OFF.

SET AMT_TRACE

Displays the contents of the environment variable.

C++ and Java

For these language bindings, there is more control over the production of trace. In each case, the AmSessionFactory has two methods that control trace:

1. setTraceLocation(location);
2. setTraceLevel(level);

The behavior of these methods matches exactly the behavior of the environment variables:

1. AMT_TRACE_PATH
2. AMT_TRACE_LEVEL

Once an AmSession has been created using an AmSessionFactory, the trace level and location are set for the complete life of that AmSession.

If set, the values of the properties in the AmSessionFactory take precedence over any AMT trace environment variables.

Example trace

The following example trace shows 'typical' trace output.

```
Trace for program d:\output\bin\amITSR.exe <<< AMT trace >>>
started at Sat Jun 12 08:28:33 1999
```

```
@(!) <<< *** Code Level is 1.0.0 *** >>>
!(03787) BuildDate Jun 11 1999
!(03787) Trace Level is 2

(03787)@08:28:33.728
-->xmq_XXXInitialize

---->ObtainSystemCP
!(03787) Code page is 437

<----ObtainSystemCP (rc = 0)

<--xmq_XXXInitialize (rc = 0)

-->amSessCreateX

---->amCheckAllBlanks()

<----amCheckAllBlanks() (rc = 0)
```

Using trace (AS/400, UNIX, and Windows)

```
---->amCheckValidName()
<----amCheckValidName() (rc = 1)
!(03787) Session name is: plenty
---->amHashTableCreate()
<----amHashTableCreate() (rc = AM_ERR_OK)
---->amSessClearErrorCodes
<----amSessClearErrorCodes (rc = 0)
...
---->amMaSrvCreate
!(03787) Service object created [9282320]
<----amMaSrvCreate (rc = AM_ERR_OK)
---->amMaSrvSetSessionHandle
!(03787) Object handle[9282320]
<----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
---->amHashTableAddHandle()
<----amHashTableAddHandle() (rc = AM_ERR_OK)
---->amMaSrvCreate
!(03787) Service object created [9285144]
<----amMaSrvCreate (rc = AM_ERR_OK)
---->amMaSrvSetSessionHandle
!(03787) Object handle[9285144]
<----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
---->amHashTableAddHandle()
<----amHashTableAddHandle() (rc = AM_ERR_OK)
(03787)08:28:33.738
---->amMaSrvCreate
!(03787) Service object created [9287968]
<----amMaSrvCreate (rc = AM_ERR_OK)
---->amMaSrvSetSessionHandle
!(03787) Object handle[9287968]
<----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
---->amHashTableAddHandle()
<----amHashTableAddHandle() (rc = AM_ERR_OK)
---->amMaSrvCreate
!(03787) Service object created [9290792]
<----amMaSrvCreate (rc = AM_ERR_OK)
---->amMaSrvSetSessionHandle
!(03787) Object handle[9290792]
<----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
```

Using trace (AS/400, UNIX, and Windows)

```
-----amHashTableAddHandle()
<-----amHashTableAddHandle() (rc = AM_ERR_OK)
-----amMaSrvCreate
!(03787) Service object created [9293616]
<-----amMaSrvCreate (rc = AM_ERR_OK)
-----amMaSrvSetSessionHandle
!(03787) Object handle[9293616]
<-----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
-----amHashTableAddHandle()
<-----amHashTableAddHandle() (rc = AM_ERR_OK)
-----amMaSrvCreate
!(03787) Service object created [9296440]
<-----amMaSrvCreate (rc = AM_ERR_OK)
-----amMaSrvSetSessionHandle
!(03787) Object handle[9296440]
<-----amMaSrvSetSessionHandle (rc = AM_ERR_OK)
-----amMaSrvSetSubReceiverHandle
!(03787) Object handle[9293616]
<-----amMaSrvSetSubReceiverHandle (rc = AM_ERR_OK)
-----amMaMsgCreate
!(03787) message object created -[10420288]
<-----amMaMsgCreate (rc = AM_ERR_OK)
-----amHashTableAddHandle()
<-----amHashTableAddHandle() (rc = AM_ERR_OK)
-----amMaMsgCreate
!(03787) message object created -[10432440]
<-----amMaMsgCreate (rc = AM_ERR_OK)
-----amHashTableAddHandle()
<-----amHashTableAddHandle() (rc = AM_ERR_OK)
-----amMaPolCreate
!(03787) policy object created.
!(03787) policy object initialized.
<-----amMaPolCreate (rc = AM_ERR_OK)
-----amHashTableAddHandle()
<-----amHashTableAddHandle() (rc = AM_ERR_OK)
-----amMaPolCreate
!(03787) policy object created.
!(03787) policy object initialized.
<-----amMaPolCreate (rc = AM_ERR_OK)
```

Using trace (AS/400, UNIX, and Windows)

```
-----amHashTableAddHandle()

<-----amHashTableAddHandle() (rc = AM_ERR_OK)

-----amMaPolSetIntProps
!(03787) Object handle[10446656]
!(03787) [AMPOL_IPR_APR_CON_CNT] set to [0x1]

(03787)@08:28:33.748
<-----amMaPolSetIntProps (rc = AM_ERR_OK)

-----amMaPolSetStringProp
!(03787) Object handle[10446656]
!(03787) [AMPOL_SPR_APR_MGR_NAME] set to [plenty]

<-----amMaPolSetStringProp (rc = AM_ERR_OK)

-----amMaPolSetStringProp
!(03787) Object handle[10446656]
!(03787) [AMPOL_SPR_APR_CON_NAME] set to [plenty]

<-----amMaPolSetStringProp (rc = AM_ERR_OK)

-----amMaSrvSetStringProp
!(03787) Object handle[9282320]
!(03787) [AMSRV_SPR_QUEUE_NAME] set to [SYSTEM.DEFAULT.SENDER]

<-----amMaSrvSetStringProp (rc = AM_ERR_OK)

-----amMaSrvSetStringProp
!(03787) Object handle[9285144]
!(03787) [AMSRV_SPR_QUEUE_NAME] set to []

<-----amMaSrvSetStringProp (rc = AM_ERR_OK)

-----amMaSrvSetStringProp
!(03787) Object handle[9287968]
!(03787) [AMSRV_SPR_QUEUE_NAME] set to [SYSTEM.DEFAULT.RECEIVER]

<-----amMaSrvSetStringProp (rc = AM_ERR_OK)

-----amMaSrvSetStringProp
!(03787) Object handle[9290792]
!(03787) [AMSRV_SPR_QUEUE_NAME] set to [SYSTEM.DEFAULT.PUBLISHER]

<-----amMaSrvSetStringProp (rc = AM_ERR_OK)

-----amMaSrvSetStringProp
!(03787) Object handle[9293616]
!(03787) [AMSRV_SPR_QUEUE_NAME] set to [SYSTEM.DEFAULT.SUBSCRIBER]

<-----amMaSrvSetStringProp (rc = AM_ERR_OK)

-----amMaPolSetIntProps
!(03787) Object handle[10451304]
!(03787) [AMPOL_IPR_SMO_SYNCPOINT] set to [0xc030003]

<-----amMaPolSetIntProps (rc = AM_ERR_OK)

-----amMaPolSetIntProps
!(03787) Object handle[10451304]
!(03787) [AMPOL_IPR_RMO_SYNCPOINT] set to [0xd060002]

<-----amMaPolSetIntProps (rc = AM_ERR_OK)

-----amActivateFiles
!(03787) No DATAPATH specified from API
```


Using trace (AS/400, UNIX, and Windows)

```
!(03787) No repository FILE specified from API
!(03787) Repository[H:\MQSeries\amt\amt.xml]
!(03787) Repository ACTIVE
!(03787) No local host FILE specified from API
!(03787) Local Host[H:\MQSeries\amt\amthost.xml]
!(03787) Local Host File ACTIVE

<----amActivateFiles (rc = 1)

---->amErrTranslate

<----amErrTranslate (rc = 0)

<--amSessCreateX (rc = 0)

...
```

Using trace (OS/390)

The AMI provides two types of trace on OS/390:

Formatted trace	Records spooled to a printer or directed to a file, which can be directly interpreted using TSO/ISPF browse, edit or print utilities.
GTF trace	Data captured on entry to and exit from high level and object level AMI function calls, which must be formatted by IPCS before viewing.

Formatted Trace

Formatted trace records are written on function entry and exit and at other points of execution where useful information can be gathered.

The format of the records is as follows:

Entry:

```
hh:mm:ss.tttt ---->function name()
```

Exit:

```
hh:mm:ss.tttt <----function name() (rc = n)
```

Data:

```
! information
```

Timestamps of entry and exit records are in local time, and are accurate to 1/10000 second. The function call depth is indicated for entry and exit records by the dashes in the '---->' or '<----' prefixes; two dashes per call level. For exit records, 'n' indicates the reason code on completion of the function. The default is to trace up to a depth of two function call levels, but this can be varied for batch applications. See "Control of formatted trace".

This is 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:

```
//JOBSTEP EXEC PGM=AMIapp,PARM='ENVAR(AMT_TRACE_LEVEL=5)'
```

For COBOL programs, Language Environment parameters are specified following the '/' delimiter. For example:

```
//JOBSTEP EXEC PGM=AMICob,PARM='/ENVAR(AMT_TRACE_LEVEL=9)'
```

Because CICS and IMS applications cannot easily set environment variables to control the trace level, the trace level defaults under CICS to a high setting. This ensures that all AMI trace points will be captured.

GTF Trace

AMI captures trace data for GTF at entry to and exit from each user-callable object level and high level AMI function. Entry trace data include function name and parameters. Exit trace data include function name and returned values.

IMS, batch, and RRS-batch AMI applications direct the trace data to GTF as user entries, using GTF event identifiers '5E9' for entry, and '5EA' for exit. These identifiers are the same as those used by 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
+0000 00F63080 C1F8E2D5 C5D3D3E2 8194E285 || .6..A8SNELLSamSe ||
+0010 A2C39385 8199C599 999699C3 968485A2 || sClearErrorCodes ||
+0020 00000000 00000000 00000000 0FA05B10 || ..... ||
          GMT-11/05/1999 14:49:51.564812 LOC-11/05/1999 14:49:51.564812
```

```
HEXFORMAT AID FF FID 00 EID E5EA
+0000 00F63080 C1F8E2D5 C5D3D3E2 8194E285 || .6..A8SNELLSamSe ||
+0010 A2C39385 8199C599 999699C3 968485A2 || sClearErrorCodes ||
+0020 00000000 00000000 00000000 00000000 || ..... ||
+0030 00000000 || .... ||
          GMT-11/05/1999 14:49:51.564906 LOC-11/05/1999 14:49:51.564906
```

AMI applications on CICS do not directly trace to GTF. AMI writes the same data to the current CICS trace destination(s) along with AMI formatted trace records. CICS tracing is controlled by the CICS administrator using the CICS-supplied transaction 'CEMT'.

Control of GTF Trace

AMI writes GTF trace records if, at AMI session start, GTF is started for the application's job name with option 'TRACE=USR'. GTF is usually started from the OS/390 operator's console using an installation defined procedure. The chapter

Using trace (OS/390)

“Using trace for problem determination” in the *MQSeries for OS/390 Problem Determination Guide* describes a typical GTF start-up prompt/reply sequence. If AMI and MQSeries GTF trace entries are to be captured to the same dataset, the job names for both the AMI application and the MQSeries queue manager must be specified.

If GTF is not started at the start of the AMI session, no GTF tracing will be performed for the remainder of the session.

When your AMI program fails

Reason Codes

When an AMI function call fails, it reports the level of the failure in the completion code of the call. AMI has three completion codes:

AMCC_OK	The call completed successfully
AMCC_WARNING	The call completed with unexpected results
AMCC_FAILED	An error occurred during processing

In the last two cases, AMI supplies a reason code that provides an explanation of the failure. A list of AMI reason codes is given in “Appendix A. Reason codes” on page 497.

In addition, if MQSeries is the reason for the failure, AMI supplies a secondary reason code. The secondary reason codes can be found in the *MQSeries Application Programming Reference* book.

First failure symptom report (AS/400, UNIX, and Windows)

A *first failure symptom* report is produced for unexpected and internal errors. This report is found in a file named `AMTnnnnn.FDC`, where `nnnnn` is the ID of the AMI process that is running at the time. You find this file in the working directory from which you started your AMI program, or in the directory specified by the path set in the `AMT_TRACE_PATH` environment variable. If you receive a first failure symptom report, you should contact IBM support personnel.

First failure symptom report (OS/390)

In the unlikely event that AMI detects an internal processing error from which no recovery is possible, the following actions are taken:

1. A dump is taken of the application’s data.
2. A first failure symptom report is produced.

Batch AMI applications write a Language Environment dump to `SYSOUT`. CICS AMI applications create a CICS transaction dump, with identifier ‘MAMT’.

Batch AMI applications write the first failure symptom report to the formatted trace data set (`AMTTRACE`), if allocated, otherwise to `SYSOUT`. CICS AMI applications write the symptom report to `SYSOUT`.

The formatted diagnostic information starts with a summary that includes:

```
Date/Time  
Code Level  
Function Name  
Probe Id          (code point within function)  
Build Date  
Major Error Code  
Minor Error Code  
Comment Lines
```

When your AMI program fails

Following the summary is a list of the stored function stack, indicating the current function call sequence. Following this is a list of the latest 40 function calls. Each item contains:

Entry/Exit indicator Function name Return Code

Other sources of information

AMI makes use of MQSeries as a transport mechanism and so MQSeries error logs and trace information can provide useful information. See the *MQSeries System Administration* manual for details of how to activate these problem determination aids.

Common causes of problems

- With the C object interface, most functions require a handle to the object they refer to. If this handle is not valid, the results are unpredictable.
- Completion code 2 (AMRC_ERROR) together with reason code 110 (AMRC_TRANSPORT_NOT_AVAILABLE) returned by **amInitialize** or **amSesOpen** (or the equivalent in COBOL, C++ and Java) normally indicates that the underlying MQSeries queue manager the AMI is attempting to use is not started (or does not exist). This might be because of a missing or incorrect xml repository file or because the data in the local host file is incorrect.
- Completion code 2 (AMRC_ERROR) together with reason code 49 (AMRC_TRANSPORT_ERR) indicates that an error was detected by the underlying MQSeries transport. The secondary reason code returned by the appropriate 'get last error' function for the object concerned will provide the related the MQSeries reason code. This error occurs most frequently during an attempt to open an underlying MQSeries queue object that does not exist (or has an incorrect type). This can be because it has never been created or because a missing or incorrect xml repository file is providing an incorrect queue name.

Part 8. Appendixes

Appendix A. Reason codes

This appendix contains a description of the AMRC_* reason codes, divided into three sections according to the value of the corresponding completion code. Within each section they are in alphabetic order. For a list of reason codes in numeric order, see “Appendix B. Constants” on page 509.

In some circumstances the AMI returns a secondary reason code that comes from MQSeries, the underlying transport layer. Please refer to the *MQSeries Application Programming Reference* manual for details of these reason codes.

Reason code: OK

The following reason code is returned with completion code: AMCC_OK

AMRC_NONE

The request was successful with no error or warning returned.

Reason code: Warning

The following reason codes are returned with completion code: AMCC_WARNING

AMRC_BACKED_OUT

The unit of work has been backed out.

AMRC_BACKOUT_LIMIT_ERR

The backout count of a received message was found to have exceeded its backout limit. The message was returned to the application and was requeued to the backout requeue queue.

AMRC_BACKOUT_REQUEUE_ERR

The backout count of a received message was found to have exceeded its backout limit. The message was returned to the application. It could not be requeued to the backout requeue queue.

AMRC_CCSID_NOT_SUPPORTED

OS/390 V2 R9 (or later) is required to enable AMI publish subscribe or message element support under CICS. Ensure that your Language Environment installation is set up to use Unicode character conversion. See “Unicode character conversion” on page 440 for more details, and see the *OS/390 C/C++ Programming Guide* for a list of the coded character sets supported under OS/390.

AMRC_CLOSE_SESSION_ERR

An error occurred while closing the session. The session is closed.

AMRC_ENCODING_INCOMPLETE

The message contains mixed values for integer, decimal, and floating point encodings, one or more of which are undefined. The encoding value returned to the application reflects only the encoding values that were defined.

AMRC_ENCODING_MIXED

The message contains mixed values for integer, decimal and floating point encodings, one or more of which conflict. An encoding value of undefined was returned to the application.

Reason code (warning)

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_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.

Reason code (warning)

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_CCSSID_ERR

The specified coded character value was not valid.

AMRC_CCSSID_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.

Reason code (failed)

AMRC_CCSID_PTR_ERR

The specified coded character set id pointer was not valid.

AMRC_COMMAND_ALREADY_EXISTS

A publish, subscribe, or unsubscribe command could not be added to the message because the message already contained a command element. If this message is generated from the high-level interface, it may mean that you have tried to use the same message name for sending and receiving publish/subscribe messages. It can also occur if the same message object is reused to send a message without being reset.

AMRC_COMMIT_INVALID

The commit request was not valid. On OS/390 under CICS, IMS, or RRS this can be due to calling the AMI commit functions rather than the transaction managers' own functions.

AMRC_CONN_NAME_NOT_FOUND

The connection name obtained from the repository was not found in the local host file.

AMRC_CORREL_ID_BUFF_LEN_ERR

The specified correlation id buffer length value was not valid.

AMRC_CORREL_ID_BUFF_PTR_ERR

The specified correlation id buffer pointer was not valid.

AMRC_CORREL_ID_LEN_ERR

The specified correlation id length value was too long.

AMRC_CORREL_ID_LEN_PTR_ERR

The specified correlation id length pointer was not valid.

AMRC_CORREL_ID_PTR_ERR

The specified correlation id pointer was not valid.

AMRC_DATA_BUFF_LEN_ERR

The specified data buffer length value was not valid.

AMRC_DATA_BUFF_PTR_ERR

The specified data buffer pointer was not valid.

AMRC_DATA_LEN_ERR

The specified data length was not valid.

AMRC_DATA_LEN_PTR_ERR

The specified data length pointer was not valid.

AMRC_DATA_OFFSET_ERR

The specified data offset value was not valid.

AMRC_DATA_OFFSET_PTR_ERR

The specified data offset pointer was not valid.

AMRC_DATA_PTR_ERR

The specified data pointer was not valid.

AMRC_DATA_SOURCE_NOT_UNIQUE

Message data for a send operation was passed in an application data buffer or a file, and was also found in the specified message object. Data to be sent can be included in an application buffer or a message object, but not both. Similarly, data can be included in a file or a message object, but not both. If data is sent in an application buffer or file, the message object can be reset first to remove existing data.

AMRC_DEFN_TYPE_ERR

The definition type defined for the service point in the repository was inconsistent with the definition type of the underlying message transport queue object when it was opened.

AMRC_DEFN_TYPE_PTR_ERR

The specified definition type pointer was not valid.

AMRC_DIST_LIST_INDEX_ERR

The specified distribution list index value was not valid.

AMRC_DIST_LIST_NOT_IN_REPOS

The definition name specified for creating a distribution list was not found in the repository. The object was not created.

AMRC_DIST_LIST_NOT_UNIQUE

The specified name could not be resolved to a unique distribution list because more than one distribution list with that name exists.

AMRC_ELEM_COUNT_PTR_ERR

The specified element count pointer was not valid.

AMRC_ELEM_INDEX_ERR

The specified element index value was not valid.

AMRC_ELEM_NAME_LEN_ERR

The specified element name length value was not valid.

AMRC_ELEM_NAME_PTR_ERR

The specified element name pointer was not valid.

AMRC_ELEM_NOT_FOUND

The specified element was not found.

AMRC_ELEM_PTR_ERR

The specified element pointer was not valid.

AMRC_ELEM_STRUC_ERR

The specified element structure was not valid. The structure id, version, or a reserved field contained an invalid value.

AMRC_ELEM_STRUC_NAME_BUFF_ERR

At least one of the name buffer (length and pointer) fields in the specified element structure was not valid.

AMRC_ELEM_STRUC_NAME_ERR

At least one of the name (length and pointer) fields in the specified element structure was not valid. Ensure that the name length, pointer, and name string are valid.

AMRC_ELEM_STRUC_TYPE_BUFF_ERR

At least one of the type buffer (length and pointer) fields in the specified element structure was not valid. Ensure that the type length, pointer and type string are valid.

AMRC_ELEM_STRUC_TYPE_ERR

At least one of the type (length and pointer) fields in the specified element structure was not valid.

AMRC_ELEM_STRUC_VALUE_BUFF_ERR

At least one of the value buffer (length and pointer) fields in the specified structure was not valid.

Reason code (failed)

AMRC_ELEM_STRUC_VALUE_ERR

At least one of the value (length and pointer) fields in the specified element structure was not valid. Ensure that the value length, pointer, and value string are valid.

AMRC_ENCODING_ERR

The specified encoding value was not valid.

AMRC_ENCODING_PTR_ERR

The specified encoding pointer was not valid.

AMRC_FILE_FORMAT_NOT_SUPPORTED

An attempt was made to send a file type that is not supported. Unsupported file types include OS/390 VSAM datasets, and OS/390 partitioned datasets (though an individual member of a PDS may be sent).

AMRC_FILE_MSG_FORMAT_ERR

When using physical mode file transfer, only two message formats are allowed: AMFMT_STRING (for text mode transfer), and AMFMT_NONE (for binary mode transfer). When using logical mode file transfer, any message format may be used for messages generated from OS/390 datasets. On other platforms and for HFS files on OS/390, only AMFMT_STRING and AMFMT_NONE can be used.

AMRC_FILE_NAME_LEN_ERR

The file name length passed in to a file transfer call was not valid.

AMRC_FILE_NAME_PTR_ERR

The file name pointer passed in to a file transfer call was not valid.

AMRC_FILE_NOT_FOUND

The file supplied on a file send call could not be opened. Check that the file exists and that the application has read access to it.

AMRC_FILE_TRANSFER_INVALID

An application running under CICS on OS/390 tried to perform a file transfer operation, which is invalid in this environment.

AMRC_FORMAT_BUFF_LEN_ERR

The specified format buffer length value was not valid.

AMRC_FORMAT_BUFF_PTR_ERR

The specified format buffer pointer was not valid.

AMRC_FORMAT_LEN_ERR

The specified message format string was too long.

AMRC_FORMAT_LEN_PTR_ERR

The specified format length pointer was not valid.

AMRC_FORMAT_PTR_ERR

The specified format pointer was not valid.

AMRC_GROUP_STATUS_ERR

The specified group status value was not valid.

AMRC_GROUP_STATUS_PTR_ERR

The specified group status pointer was not valid.

AMRC_HEADER_INVALID

The RFH header structure of the message was not valid.

AMRC_HEADER_TRUNCATED

The RFH header of the message was truncated.

AMRC_HOST_CACHE_ERR

A module was loaded for use as a repository file cache, but the module does not appear to be a valid repository cache.

AMRC_HOST_FILE_ERR

The contents of the local host file are not valid.

AMRC_HOST_FILENAME_ERR

The local host file name was not valid. The value of the appropriate environment variable should be corrected.

AMRC_HOST_FILE_NOT_FOUND

A local host file with the specified name was not found.

AMRC_INCOMPLETE_GROUP

The specified request failed because an attempt was made to send a message that was not in a group when the existing message group was incomplete.

AMRC_INSUFFICIENT_MEMORY

There was not enough memory available to complete the requested operation.

AMRC_INVALID_DIST_LIST_NAME

The specified distribution list name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_IF_SERVICE_OPEN

The receiver queue name could not be set because the receiver or subscriber service was open.

AMRC_INVALID_MSG_NAME

The specified message name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_POLICY_NAME

The specified policy name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_PUBLISHER_NAME

The specified publisher service name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_Q_NAME

The specified queue name was too long, or contained invalid characters.

AMRC_INVALID_RECEIVER_NAME

The specified receiver service name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_SENDER_NAME

The specified sender service name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_SESSION_NAME

The specified session name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_SUBSCRIBER_NAME

The specified subscriber service name was too long, contained invalid characters, or used the reserved prefix 'SYSTEM.'.

AMRC_INVALID_TRACE_LEVEL

A specified trace level was not valid.

Reason code (failed)

AMRC_JAVA_CLASS_ERR

A class referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_CREATE_ERR

An unexpected error occurred when creating an AMI Java object. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_FIELD_ERR

A field referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_JNI_ERR

An unexpected error occurred when calling the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_METHOD_ERR

A method referenced in AMI Java code cannot be found in the AMI Java native library. This is probably due to an incompatibility between the AMI class files and the AMI Java library. (Not applicable to the C and C++ programming languages).

AMRC_JAVA_NULL_PARM_ERR

The AMI Java code detected a null parameter that is not valid. (Not applicable to the C and C++ programming languages).

AMRC_MSG_HANDLE_ERR

The specified message handle was not valid.

AMRC_MSG_ID_BUFF_LEN_ERR

The specified message id buffer length value was not valid.

AMRC_MSG_ID_BUFF_PTR_ERR

The specified message id buffer pointer was not valid.

AMRC_MSG_ID_LEN_ERR

The specified message id length value was not valid.

AMRC_MSG_ID_LEN_PTR_ERR

The specified message id length pointer was not valid.

AMRC_MSG_ID_PTR_ERR

The specified message id pointer was not valid.

AMRC_MSG_NOT_FOUND

The specified message was not found, so the request was not carried out.

AMRC_MSG_NOT_UNIQUE

The specified name could not be resolved to a unique message because more than one message object with that name exists.

AMRC_MSG_TYPE_NOT_REPORT

The message is not a report message.

AMRC_MSG_TYPE_PTR_ERR

The specified message type pointer was not valid.

AMRC_NAME_BUFF_LEN_ERR

The specified name buffer length value was not valid.

AMRC_NAME_BUFF_PTR_ERR

The specified name buffer pointer was not valid.

AMRC_NAME_LEN_PTR_ERR

The specified name length pointer was not valid.

AMRC_NO_MSG_AVAILABLE

No message was available for a receive request after the specified wait time.

AMRC_NO_RESP_SERVICE

The publish request was not successful because a response receiver service is required for registration and was not specified.

AMRC_NOT_AUTHORIZED

The user is not authorized by the underlying transport layer to perform the specified request.

AMRC_POLICY_HANDLE_ERR

The specified policy handle was not valid.

AMRC_POLICY_NOT_FOUND

The specified policy was not found, so the request was not carried out.

AMRC_POLICY_NOT_UNIQUE

The specified name could not be resolved to a unique policy because more than one policy with that name exists.

AMRC_PRIMARY_HANDLE_ERR

The primary handle (that is, the first parameter) passed on the API call was not valid. The most probable reason for failure is that the handle passed is a synonym handle, which is not valid as the *primary* handle on any call to the AMI.

AMRC_PUBLISHER_NOT_UNIQUE

The specified name could not be resolved to a unique publisher because more than one publisher object with that name exists.

AMRC_Q_NAME_BUFF_LEN_ERR

The specified queue name buffer length value was not valid.

AMRC_Q_NAME_BUFF_PTR_ERR

The specified queue name buffer pointer was not valid.

AMRC_Q_NAME_LEN_ERR

The specified queue name length value was not valid.

AMRC_Q_NAME_LEN_PTR_ERR

The specified queue name length pointer was not valid.

AMRC_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.

Reason code (failed)

AMRC_RECEIVE_DISABLED

The specified request could not be performed because the service in the underlying transport layer is not enabled for receive requests.

AMRC_RECEIVER_NOT_UNIQUE

The specified name could not be resolved to a unique receiver because more than one receiver object with that name exists.

AMRC_REPORT_CODE_PTR_ERR

The specified report code pointer was not valid.

AMRC_REPOS_CACHE_ERR

A module was loaded for use as a host file cache, but the module does not appear to be a valid host cache.

AMRC_REPOS_ERR

An error was returned when initializing or accessing the repository. This can occur for any of the following reasons:

- The repository XML file (for instance, `amt.xml`) contains data that is not valid.
- The DTD file (`amt.dtd`) was not found or contains data that is not valid.
- The files needed to initialize the repository (located in directories `intlFiles` and `locales`) could not be located.

Check that the DTD and XML files are valid and correctly located, and that the path settings for the local host and repository files are correct.

AMRC_REPOS_FILENAME_ERR

The repository file name was not valid. The value of the appropriate environment variable should be corrected.

AMRC_REPOS_NOT_FOUND

The repository file was not found. The value of the appropriate environment variable should be corrected.

AMRC_RESERVED_NAME_IN_REPOS

The name specified for creating an object was found in the repository and is a reserved name that is not valid in a repository. The specified object was not created.

AMRC_RESP_RECEIVER_HANDLE_ERR

The response receiver service handle specified when sending a request message was not valid.

AMRC_RESP_SENDER_HANDLE_ERR

The response sender service handle specified when receiving a request message was not valid.

AMRC_RFH_ALREADY_EXISTS

A publish, subscribe, or unsubscribe command could not be added to the message because the message already contained an RFH header. The message requires a reset first, to remove existing data.

AMRC_SEND_DATA_PTR_ERR

The buffer pointer specified for sending data was not valid.

AMRC_SEND_DATA_LEN_ERR

The data length specified for sending data was not valid.

AMRC_SEND_DISABLED

The specified request could not be performed because the service in the underlying transport layer is not enabled for send requests.

AMRC_SENDER_COUNT_PTR_ERR

The specified distribution list sender count pointer was not valid.

AMRC_SENDER_NOT_UNIQUE

The specified name could not be resolved to a unique sender because more than one sender object with that name exists.

AMRC_SENDER_USAGE_ERR

The specified sender service definition type was not valid for sending responses. To be valid for sending a response, a sender service must not have a repository definition, must have been specified as a response service when receiving a previous request message and must not have been used for any purpose other than sending responses.

AMRC_SERVICE_ALREADY_CLOSED

The specified (sender, receiver, distribution list, publisher or subscriber) service was already closed.

AMRC_SERVICE_ALREADY_OPEN

The specified (sender, receiver, distribution list, publisher or subscriber) service was already open.

AMRC_SERVICE_FULL

The specified request could not be performed because the service in the underlying transport has reached its maximum message limit.

AMRC_SERVICE_HANDLE_ERR

The service handle specified for a sender, receiver, distribution list, publisher, or subscriber was not valid.

AMRC_SERVICE_NOT_FOUND

The specified (sender, receiver, distribution list, publisher, or subscriber) service was not found, so the request was not carried out.

AMRC_SERVICE_NOT_OPEN

The request failed because the specified (sender, receiver, distribution list, publisher or subscriber) service was not open.

AMRC_SESSION_ALREADY_CLOSED

The session was already closed (or terminated).

AMRC_SESSION_ALREADY_OPEN

The session was already open (or initialized).

AMRC_SESSION_EXPIRED

Under the IMS environment, the current session has been marked as expired. See "Writing IMS applications using AMI" on page 421 for an explanation of why a session may be expired. Delete the current session and create new one for the duration of this transaction.

AMRC_SESSION_HANDLE_ERR

The specified session handle was not valid.

AMRC_SESSION_NOT_OPEN

The request failed because the session was not open.

AMRC_SUBSCRIBER_NOT_UNIQUE

The specified name could not be resolved to a unique subscriber because more than one subscriber object with that name exists.

AMRC_TRANSPORT_ERR

An error was reported by the underlying (MQSeries) message transport layer. The message transport reason code can be obtained by the secondary reason

Reason code (failed)

code value returned from a 'GetLastError' request for the AMI object concerned. For more information, see "Common causes of problems" on page 493.

AMRC_TRANSPORT_LIBRARY_ERR

An error occurred loading the transport library.

AMRC_TRANSPORT_NOT_AVAILABLE

The underlying transport layer is not available.

AMRC_UNEXPECTED_ERR

An unexpected error occurred.

AMRC_WAIT_TIME_ERR

The specified wait-time value was not valid.

AMRC_WAIT_TIME_PTR_ERR

The specified wait time pointer was not valid.

AMRC_WAIT_TIME_READ_ONLY

An attempt was made to set the wait time in a policy object for which the wait-time was read-only.

Appendix B. Constants

This appendix lists the values of the named constants used by the functions described in this manual. For information about MQSeries constants not in this list, see the *MQSeries Application Programming Reference* manual and the *MQSeries Programmable System Management* manual.

The constants

The constants are grouped according to the parameter or field to which they relate. Names of the constants in a group begin with a common prefix of the form AMxxxx_, where xxxx represents a string of 0 through 4 characters that indicates the nature of the values defined in that group. Within each group, constants are listed in numeric (or alphabetic) order.

Character strings are shown delimited by double quotation marks; the quotation marks are not part of the value.

AMB (Boolean constants)

AMB_FALSE	0L
AMB_TRUE	1L

AMBRW (Browse constants)

AMBRW_UNLOCK	1L
AMBRW_LOCK	2L
AMBRW_FIRST	4L
AMBRW_NEXT	8L
AMBRW_CURRENT	16L
AMBRW_RECEIVE_CURRENT	32L
AMBRW_DEFAULT	AMBRW_NEXT
AMBRW_LOCK_NEXT	(AMBRW_LOCK + AMBRW_NEXT)
AMBRW_LOCK_FIRST	(AMBRW_LOCK + AMBRW_FIRST)
AMBRW_LOCK_CURRENT	(AMBRW_LOCK + AMBRW_CURRENT)

AMCC (Completion codes)

AMCC_OK	0L
AMCC_WARNING	1L
AMCC_FAILED	2L

AMDEF (Service and policy definitions)

AMDEF_POL	"AMT.SYSTEM.POLICY"
AMDEF_PUB	"AMT.SYSTEM.PUBLISHER"
AMDEF_RCV	"AMT.SYSTEM.RECEIVER"
AMDEF_RSP_SND	"AMT.SYSTEM.RESPONSE.SENDER"
AMDEF_SND	"AMT.SYSTEM.SENDER"
AMDEF_SUB	"AMT.SYSTEM.SUBSCRIBER"
AMDEF_SYNC_POINT_POL	"AMT.SYSTEM.SYNCPOINT.POLICY"

AMDT (Definition type constants)

AMDT_UNDEFINED	0L
AMDT_TEMP_DYNAMIC	2L
AMDT_DYNAMIC	3L
AMDT_PREDEFINED	4L

Constants

AMENC (Encoding constants)

AMENC_NORMAL	0L
AMENC_REVERSED	1L
AMENC_NORMAL_FLOAT_390	2L
AMENC_REVERSED_FLOAT_390	3L
AMENC_UNDEFINED	4L
AMENC_NATIVE	AMENC_NORMAL (UNIX)
AMENC_NATIVE	AMENC_REVERSED (WIN32)
AMENC_NATIVE	AMENC_NORMAL_FLOAT_390 (OS/390)

AMFB (Feedback codes)

AMFB_NONE	0L
AMFB_EXPIRATION	1L
AMFB_COA	2L
AMFB_COD	3L
AMFB_ERROR	-1L

AMFMT (Format constants)

AMFMT_NONE	" "
AMFMT_RF_HEADER	"MQHRF "
AMFMT_STRING	"MQSTR "
AMFMT_RF2_HEADER	"MQHRF2 "

AMGF and AMGRP (Group status constants)

AMGF_IN_GROUP	1L
AMGF_FIRST	2L
AMGF_LAST	4L
AMGRP_MSG_NOT_IN_GROUP	0L
AMGRP_FIRST_MSG_IN_GROUP	(AMGF_IN_GROUP AMGF_FIRST)
AMGRP_MIDDLE_MSG_IN_GROUP	AMGF_IN_GROUP
AMGRP_LAST_MSG_IN_GROUP	(AMGF_IN_GROUP AMGF_LAST)
AMGRP_ONLY_MSG_IN_GROUP	(AMGF_IN_GROUP AMGF_FIRST AMGF_LAST)

AMH (Handle constants)

AMH_NULL_HANDLE	(AMHANDLE) 0L
AMH_INVALID_HANDLE	(AMHANDLE) -1L

AMLEN (String length constants)

AMLEN_NULL_TERM	-1L
AMLEN_MAX_NAME_LENGTH	256L

AMMCD (Message Content Descriptor tag names)

AMMCD_MSG_SERVICE_DOMAIN	"mcd.Msd"
AMMCD_MSG_SET	"mcd.Set"
AMMCD_MSG_TYPE	"mcd.Type"
AMMCD_MSG_FORMAT	"mcd.Fmt"

AMMT (Message types)

AMMT_REQUEST	1L
AMMT_REPLY	2L
AMMT_REPORT	4L
AMMT_DATAGRAM	8L

AMPS (Publish/subscribe)

Publish/Subscribe constants

Publish/subscribe tag names

AMPS_COMMAND	"MQPSCommand"
AMPS_COMP_CODE	"MQPSCompCode"
AMPS_DELETE_OPTIONS	"MQPSDe10pts"
AMPS_ERROR_ID	"MQPSErrorId"
AMPS_ERROR_POS	"MQPSErrorPos"
AMPS_PARAMETER_ID	"MQPSParmId"
AMPS_PUBLICATION_OPTIONS	"MQPSPub0pts"
AMPS_TIMESTAMP	"MQPSPubTime"
AMPS_Q_MGR_NAME	"MQPSQMgrName"
AMPS_Q_NAME	"MQPSQName"
AMPS_REASON	"MQPSReason"
AMPS_REASON_TEXT	"MQPSReasonText"
AMPS_REGISTRATION_OPTIONS	"MQPSReg0pts"
AMPS_SEQUENCE_NUMBER	"MQPSSeqNum"
AMPS_STREAM_NAME	"MQPSStreamName"
AMPS_STRING_DATA	"MQPSStringData"
AMPS_TOPIC	"MQPSTopic"
AMPS_USER_ID	"MQPSUserId"
AMPS_FILTER	"MQPSFilter"
AMPS_SUBSCRIPTION_POINT	"MQPSSubPoint"
AMPS_SEQUENCE	"MQPSSequence"
AMPS_CONTROL	"MQPSControl"

Publish/subscribe tag values

AMPS_ANONYMOUS	"Anon"
AMPS_CORREL_ID_AS_ID	"CorrelAsId"
AMPS_DEREGISTER_ALL	"DeregAll"
AMPS_DIRECT_REQUESTS	"DirectReq"
AMPS_INCLUDE_STREAM_NAME	"InclStreamName"
AMPS_INFORM_IF_RETAINED	"InformIfRet"
AMPS_LOCAL	"Local"
AMPS_NEW_PUBS_ONLY	"NewPubsOnly"
AMPS_PUB_ON_REQUEST_ONLY	"PubOnReqOnly"
AMPS_DELETE_PUBLICATION	"DeletePub"
AMPS_DEREGISTER_PUBLISHER	"DeregPub"
AMPS_DEREGISTER_SUBSCRIBER	"DeregSub"
AMPS_PUBLISH	"Publish"
AMPS_REGISTER_PUBLISHER	"RegPub"
AMPS_REGISTER_SUBSCRIBER	"RegSub"
AMPS_REQUEST_UPDATE	"ReqUpdate"
AMPS_IS_RETAINED_PUBLICATION	"IsRetainedPub"
AMPS_NO_REGISTRATION	"NoReg"
AMPS_NONE	"None"
AMPS_OTHER_SUBSCRIBERS_ONLY	"OtherSubsOnly"
AMPS_RETAIN_PUBLICATION	"RetainPub"
AMPS_PERSISTENT	"Pers"
AMPS_NON_PERSISTENT	"NonPers"
AMPS_PERSISTENT_AS_PUBLISHER	"PersAsPub"
AMPS_PERSISTENT_AS_QUEUE	"PersAsQueue"
AMPS_CC_OK	"0"
AMPS_CC_WARNING	"1"
AMPS_CC_ERROR	"2"

Other publish/subscribe constants

AMPS_APPL_TYPE	"OPT_APP_GRP "
AMPS_MSG_TYPE	"OPT_MSG_TYPE "

Constants

AMRC (Reason codes)

AMRC_NONE	0
AMRC_UNEXPECTED_ERR	1
AMRC_INVALID_Q_NAME	2
AMRC_INVALID_SENDER_NAME	3
AMRC_INVALID_RECEIVER_NAME	4
AMRC_INVALID_PUBLISHER_NAME	5
AMRC_INVALID_SUBSCRIBER_NAME	6
AMRC_INVALID_POLICY_NAME	7
AMRC_INVALID_MSG_NAME	8
AMRC_INVALID_SESSION_NAME	9
AMRC_INVALID_DIST_LIST_NAME	10
AMRC_POLICY_HANDLE_ERR	11
AMRC_SERVICE_HANDLE_ERR	12
AMRC_MSG_HANDLE_ERR	13
AMRC_SESSION_HANDLE_ERR	14
AMRC_BROWSE_OPTIONS_ERR	15
AMRC_INSUFFICIENT_MEMORY	16
AMRC_WAIT_TIME_READ_ONLY	17
AMRC_SERVICE_NOT_FOUND	18
AMRC_MSG_NOT_FOUND	19
AMRC_POLICY_NOT_FOUND	20
AMRC_SENDER_NOT_UNIQUE	21
AMRC_RECEIVER_NOT_UNIQUE	22
AMRC_PUBLISHER_NOT_UNIQUE	23
AMRC_SUBSCRIBER_NOT_UNIQUE	24
AMRC_MSG_NOT_UNIQUE	25
AMRC_POLICY_NOT_UNIQUE	26
AMRC_DIST_LIST_NOT_UNIQUE	27
AMRC_RECEIVE_BUFF_PTR_ERR	28
AMRC_RECEIVE_BUFF_LEN_ERR	29
AMRC_SEND_DATA_PTR_ERR	30
AMRC_SEND_DATA_LEN_ERR	31
AMRC_INVALID_IF_SERVICE_OPEN	32
AMRC_SERVICE_ALREADY_OPEN	33
AMRC_DATA_SOURCE_NOT_UNIQUE	34
AMRC_NO_MSG_AVAILABLE	35
AMRC_SESSION_ALREADY_OPEN	36
AMRC_SESSION_ALREADY_CLOSED	37
AMRC_ELEM_NOT_FOUND	38
AMRC_ELEM_COUNT_PTR_ERR	39
AMRC_ELEM_NAME_PTR_ERR	40
AMRC_ELEM_NAME_LEN_ERR	41
AMRC_ELEM_INDEX_ERR	42
AMRC_ELEM_PTR_ERR	43
AMRC_ELEM_STRUC_ERR	44
AMRC_ELEM_STRUC_NAME_ERR	45
AMRC_ELEM_STRUC_VALUE_ERR	46
AMRC_ELEM_STRUC_NAME_BUFF_ERR	47
AMRC_ELEM_STRUC_VALUE_BUFF_ERR	48
AMRC_TRANSPORT_ERR	49
AMRC_TRANSPORT_WARNING	50
AMRC_ENCODING_INCOMPLETE	51
AMRC_ENCODING_MIXED	52
AMRC_ENCODING_ERR	53
AMRC_BEGIN_INVALID	54
AMRC_NO_REPLY_TO_INFO	55
AMRC_SERVICE_ALREADY_CLOSED	56

Constants

AMRC_SESSION_NOT_OPEN	57
AMRC_DIST_LIST_INDEX_ERR	58
AMRC_WAIT_TIME_ERR	59
AMRC_SERVICE_NOT_OPEN	60
AMRC_HEADER_TRUNCATED	61
AMRC_HEADER_INVALID	62
AMRC_DATA_LEN_ERR	63
AMRC_BACKOUT_REQUEUE_ERR	64
AMRC_BACKOUT_LIMIT_ERR	65
AMRC_COMMAND_ALREADY_EXISTS	66
AMRC_UNEXPECTED_RECEIVE_ERR	67
AMRC_UNEXPECTED_SEND_ERR	68
AMRC_SENDER_USAGE_ERR	70
AMRC_MSG_TRUNCATED	71
AMRC_CLOSE_SESSION_ERR	72
AMRC_READ_OFFSET_ERR	73
AMRC_RFH_ALREADY_EXISTS	74
AMRC_GROUP_STATUS_ERR	75
AMRC_MSG_ID_LEN_ERR	76
AMRC_MSG_ID_PTR_ERR	77
AMRC_MSG_ID_BUFF_LEN_ERR	78
AMRC_MSG_ID_BUFF_PTR_ERR	79
AMRC_MSG_ID_LEN_PTR_ERR	80
AMRC_CORREL_ID_LEN_ERR	81
AMRC_CORREL_ID_PTR_ERR	82
AMRC_CORREL_ID_BUFF_LEN_ERR	83
AMRC_CORREL_ID_BUFF_PTR_ERR	84
AMRC_CORREL_ID_LEN_PTR_ERR	85
AMRC_FORMAT_LEN_ERR	86
AMRC_FORMAT_PTR_ERR	87
AMRC_FORMAT_BUFF_PTR_ERR	88
AMRC_FORMAT_LEN_PTR_ERR	89
AMRC_FORMAT_BUFF_LEN_ERR	90
AMRC_NAME_BUFF_PTR_ERR	91
AMRC_NAME_LEN_PTR_ERR	92
AMRC_NAME_BUFF_LEN_ERR	93
AMRC_Q_NAME_LEN_ERR	94
AMRC_Q_NAME_PTR_ERR	95
AMRC_Q_NAME_BUFF_PTR_ERR	96
AMRC_Q_NAME_LEN_PTR_ERR	97
AMRC_Q_NAME_BUFF_LEN_ERR	98
AMRC_WAIT_TIME_PTR_ERR	99
AMRC_CCSID_PTR_ERR	100
AMRC_ENCODING_PTR_ERR	101
AMRC_DEFN_TYPE_PTR_ERR	102
AMRC_CCSID_ERR	103
AMRC_DATA_LEN_PTR_ERR	104
AMRC_GROUP_STATUS_PTR_ERR	105
AMRC_DATA_OFFSET_PTR_ERR	106
AMRC_RESP_SENDER_HANDLE_ERR	107
AMRC_RESP_RECEIVER_HANDLE_ERR	108
AMRC_NOT_AUTHORIZED	109
AMRC_TRANSPORT_NOT_AVAILABLE	110
AMRC_BACKED_OUT	111
AMRC_INCOMPLETE_GROUP	112
AMRC_SEND_DISABLED	113
AMRC_SERVICE_FULL	114
AMRC_NOT_CONVERTED	115
AMRC_RECEIVE_DISABLED	116

Constants

AMRC_GROUP_BACKOUT_LIMIT_ERR	117
AMRC_SENDR_COUNT_PTR_ERR	118
AMRC_MULTIPLE_REASONS	119
AMRC_NO_RESP_SERVICE	120
AMRC_DATA_PTR_ERR	121
AMRC_DATA_BUFF_LEN_ERR	122
AMRC_DATA_BUFF_PTR_ERR	123
AMRC_DEFN_TYPE_ERR	124
AMRC_BACKOUT_INVALID	125
AMRC_COMMIT_INVALID	126
AMRC_DATA_OFFSET_ERR	127
AMRC_FILE_SYSTEM_ERR	128
AMRC_FILE_ALREADY_EXISTS	129
AMRC_REPORT_CODE_PTR_ERR	130
AMRC_MSG_TYPE_PTR_ERR	131
AMRC_FILE_FORMAT_CONVERTED	132
AMRC_FILE_TRUNCATED	133
AMRC_FILE_NOT_FOUND	134
AMRC_NOT_A_FILE	135
AMRC_FILE_NAME_LEN_ERR	136
AMRC_FILE_NAME_PTR_ERR	137
AMRC_RFH2_FORMAT_ERR	138
AMRC_CCSID_NOT_SUPPORTED	139
AMRC_FILE_MSG_FORMAT_ERR	140
AMRC_MSG_TYPE_NOT_REPORT	141
AMRC_ELEM_STRUC_TYPE_ERR	142
AMRC_ELEM_STRUC_TYPE_BUFF_ERR	143
AMRC_FILE_TRANSFER_INVALID	144
AMRC_FILE_NOT_WRITTEN	145
AMRC_FILE_FORMAT_NOT_SUPPORTED	146
AMRC_INVALID_TRACE_LEVEL	400
AMRC_CONN_NAME_NOT_FOUND	401
AMRC_HOST_FILE_NOT_FOUND	402
AMRC_HOST_FILENAME_ERR	403
AMRC_HOST_FILE_ERR	404
AMRC_POLICY_NOT_IN_REPOS	405
AMRC_SENDR_NOT_IN_REPOS	406
AMRC_RECEIVER_NOT_IN_REPOS	407
AMRC_DIST_LIST_NOT_IN_REPOS	408
AMRC_PUBLISHER_NOT_IN_REPOS	409
AMRC_SUBSCRIBER_NOT_IN_REPOS	410
AMRC_RESERVED_NAME_IN_REPOS	411
AMRC_REPOS_FILENAME_ERR	414
AMRC_REPOS_WARNING	415
AMRC_REPOS_ERR	416
AMRC_REPOS_NOT_FOUND	418
AMRC_TRANSPORT_LIBRARY_ERR	419
AMRC_HOST_CACHE_ERR	420
AMRC_REPOS_CACHE_ERR	421
AMRC_PRIMARY_HANDLE_ERR	422
AMRC_SESSION_EXPIRED	423
AMRC_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

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.SYNCPPOINT.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
----------------	-----

Constants

Appendix C. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Notices

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX	AS/400	BookManager
CICS	DB2	IBM
IBMLink	Language Environment	MQSeries
RACF	OS/390	OS/400
VisualAge	VSE/ESA	

Java, JDK and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, Visual C++ and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, or service names may be the trademarks or service marks of others.

Glossary of terms and abbreviations

This glossary defines terms and abbreviations used in this book. If you do not find the term you are looking for, see the Index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

B

broker. See *message broker*.

C

connection. An AMI connection maps a logical queue manager name in a policy to a real queue manager name. This allows applications running on different nodes to use the same policy to connect to different queue managers.

correlation identifier. This is used as a key to a message, for example to correlate a response message with a request message. The AMI normally sets this in a response message by copying the message identifier from the request message. See also *request/response* and *selection message*.

D

datagram. The simplest message that MQSeries supports. Also known as *send-and-forget*. This type of message does not require a reply. Compare with *request/response*.

distribution list. An AMI service. It contains a list of sender services, enabling a message to be sent to multiple destinations in one operation.

E

Extensible Markup Language (XML). A W3C standard for the representation of data.

F

filter. An expression that is applied to the content of a message to determine how the message is to be processed. See also *subscription filter*.

L

local host file. Defines the mapping from a logical connection name to a real MQSeries queue manager on the local machine.

M

message. A message defines what is sent from one program to another in an AMI application. See also *service* and *policy*.

message broker. A set of execution processes hosting one or more message flows.

message descriptor (MQMD). Control information describing the message format and properties that is carried as part of an MQSeries message.

message identifier. An identifier for the message. It is usually unique, and typically it is generated by the message transport (MQSeries).

message object. An AMI object. It contains attributes of the message, such as the message identifier and correlation identifier, and options that are used when sending or receiving the message (most of which come from the policy definition). It can also contain the message data.

message queue. See *queue*.

message queue interface (MQI). The programming interface provided by MQSeries queue managers. It allows application programs to access message queuing services. The AMI provides a simpler interface to these services.

MQRFH header. Header added to an MQSeries message to carry control information, typically for use by a broker (for example, in a publish/subscribe system).

P

point-to-point. Style of messaging application in which the sending application knows the destination of the message. Compare with *publish/subscribe*.

policy. A policy defines how a message is sent in an AMI application. It encapsulates many of the options available in the MQI. Its definition can be stored in a repository. See also *service*.

publish/subscribe. Style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers) using a broker. Compare with *point-to-point*. See also *topic*.

publisher. (1) An AMI service. It contains a sender service where the destination is a publish/subscribe

Glossary

broker. (2) An application that makes information about a specified topic available to a broker in a publish/subscribe system.

Q

queue. An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages: they point to other queues, or can be used as models for dynamic queues.

queue manager. A system program that provides queuing services to applications. It provides an application programming interface (the MQI) so that programs can access messages on the queues that the queue manager owns.

R

receiver. An AMI service. It represents a source (such as an MQSeries queue) from which messages are received. Its definition is stored in a repository as a service point.

repository. A repository provides definitions for services and policies. If the name of a service or policy is not found in the repository, or an AMI application does not have a repository, the definitions built into the AMI are used. See also *repository file*.

repository file. File that stores repository definitions in XML (Extensible Markup Language) format.

request/response. Type of messaging application in which a request message is used to request a response from another application. Compare with *datagram*. See also *response sender* and *selection message*.

response sender. A special type of sender service that is used to send a response to a request message. It must use the definition built into the AMI, so it must not be defined in the repository.

S

selection message. A message object that is used to selectively receive a message by specifying its correlation identifier. Used in request/response messaging to correlate a response message with its request message.

send-and-forget. See *datagram*.

sender. An AMI service. It represents a destination (such as an MQSeries queue) to which messages are sent. Its definition is stored in a repository as a service point.

service. A service defines where a message is sent in an AMI application. Senders, receivers, distribution lists, publishers, and subscribers are all types of service. Their definitions can be stored in a repository. See also *policy*.

service point. The definition in a repository of a sender or receiver service.

session. An AMI object. It creates and manages all other AMI objects (message, service, policy and connection objects), and it provides the scope for a unit of work when transactional processing is used.

subscriber. (1) An AMI service. It contains a sender service to send subscribe and unsubscribe messages to a publish/subscribe broker, and a receiver service to receive publications from the broker. (2) An application that requests information about a specified topic from a publish/subscribe broker.

subscription filter. A predicate that specifies a subset of messages to be delivered to a particular subscriber.

T

topic. A character string that describes the nature of the data that is being published in a publish/subscribe system.

W

W3C. World Wide Web Consortium. An international industry consortium set up to develop common protocols to promote evolution and interoperability of the World Wide Web.

X

XML. Extensible Markup Language.

Bibliography

This section describes the documentation available for all current MQSeries products.

MQSeries cross-platform publications

Most of these publications, which are sometimes referred to as the MQSeries “family” books, apply to all MQSeries Level 2 products. The latest MQSeries Level 2 products are:

- | • MQSeries for AIX, V5.2
- | • MQSeries for AS/400, V5.2
- | • MQSeries for AT&T GIS UNIX, V2.2
- | • MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1
- | • MQSeries for Compaq Tru64 UNIX, V5.1
- | • MQSeries for HP-UX, V5.2
- | • MQSeries for Linux, V5.2
- | • MQSeries for OS/2 Warp, V5.1
- | • MQSeries for OS/390, V5.2
- | • MQSeries for SINIX and DC/OSx, V2.2
- | • MQSeries for Sun Solaris, V5.2
- | • MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- | • MQSeries for Tandem NonStop Kernel, V2.2.0.1
- | • MQSeries for VSE/ESA, V2.1.1
- | • MQSeries for Windows, V2.0
- | • MQSeries for Windows, V2.1
- | • MQSeries for Windows NT and Windows 2000, V5.2

The MQSeries cross-platform publications are:

- *MQSeries Brochure*, G511-1908
- *An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries Queue Manager Clusters*, SC34-5349
- *MQSeries Clients*, GC33-1632
- *MQSeries System Administration*, SC33-1873
- *MQSeries MQSC Command Reference*, SC33-1369
- *MQSeries Event Monitoring*, SC34-5760
- *MQSeries Programmable System Management*, SC33-1482
- *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390
- *MQSeries Messages*, GC33-1876
- *MQSeries Application Programming Guide*, SC33-0807

- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Programming Interfaces Reference Summary*, SX33-6095
- *MQSeries Using C++*, SC33-1877
- *MQSeries Using Java*, SC34-5456
- *MQSeries Application Messaging Interface*, SC34-5604

MQSeries platform-specific publications

Each MQSeries product is documented in at least one platform-specific publication, in addition to the MQSeries family books.

MQSeries for AIX, V5.2

- | *MQSeries for AIX Quick Beginnings*, GC33-1867

MQSeries for AS/400, V5.2

- | *MQSeries for AS/400 Quick Beginnings*, GC34-5557
- | *MQSeries for AS/400 System Administration*, SC34-5558
- | *MQSeries for AS/400 Application Programming Reference (ILE RPG)*, SC34-5559

MQSeries for AT&T GIS UNIX, V2.2

- | *MQSeries for AT&T GIS UNIX System Management Guide*, SC33-1642

MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1

- | *MQSeries for Compaq (DIGITAL) OpenVMS System Management Guide*, GC33-1791

MQSeries for Compaq Tru64 UNIX, V5.1

- | *MQSeries for Compaq Tru64 UNIX Quick Beginnings*, GC34-5684

MQSeries for HP-UX, V5.2

- | *MQSeries for HP-UX Quick Beginnings*, GC33-1869

MQSeries for Linux, V5.2

- | *MQSeries for Linux Quick Beginnings*, GC34-5691

Bibliography

MQSeries for OS/2 Warp, V5.1

MQSeries for OS/2 Warp Quick Beginnings, GC33-1868

MQSeries for OS/390, V5.2

MQSeries for OS/390 Concepts and Planning Guide, GC34-5650

MQSeries for OS/390 System Setup Guide, SC34-5651

MQSeries for OS/390 System Administration Guide, SC34-5652

MQSeries for OS/390 Problem Determination Guide, GC34-5892

MQSeries for OS/390 Messages and Codes, GC34-5891

MQSeries for OS/390 Licensed Program Specifications, GC34-5893

MQSeries for OS/390 Program Directory

MQSeries link for R/3, Version 1.2

MQSeries link for R/3 User's Guide, GC33-1934

MQSeries for SINIX and DC/OSx, V2.2

MQSeries for SINIX and DC/OSx System Management Guide, GC33-1768

MQSeries for Sun Solaris, V5.2

MQSeries for Sun Solaris Quick Beginnings, GC33-1870

MQSeries for Sun Solaris, Intel Platform Edition, V5.1

MQSeries for Sun Solaris, Intel Platform Edition Quick Beginnings, GC34-5851

MQSeries for Tandem NonStop Kernel, V2.2.0.1

MQSeries for Tandem NonStop Kernel System Management Guide, GC33-1893

MQSeries for VSE/ESA, V2.1.1

MQSeries for VSE/ESA™ Licensed Program Specifications, GC34-5365

MQSeries for VSE/ESA System Management Guide, GC34-5364

MQSeries for Windows, V2.0

MQSeries for Windows User's Guide, GC33-1822

MQSeries for Windows, V2.1

MQSeries for Windows User's Guide, GC33-1965

MQSeries for Windows NT and Windows 2000, V5.2

MQSeries for Windows NT Quick Beginnings, GC34-5389

MQSeries for Windows NT Using the Component Object Model Interface, SC34-5387

MQSeries LotusScript Extension, SC34-5404

Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

HTML format

Relevant MQSeries documentation is provided in HTML format with these MQSeries products:

- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Windows NT and Windows 2000, V5.2 (compiled HTML)
- MQSeries link for R/3, V1.2

The MQSeries books are also available in HTML format from the MQSeries product family Web site at:

<http://www.ibm.com/software/mqseries/>

Portable Document Format (PDF)

PDF files can be viewed and printed using the Adobe Acrobat Reader.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. Web site at:

<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/mqseries/>

BookManager[®] format

The MQSeries library is supplied in IBM BookManager format on a variety of online library collection kits, including the *Transaction Processing and Data* collection kit, SK2T-0730. You can view the softcopy books in IBM BookManager format using the following IBM licensed programs:

- BookManager READ/2
- BookManager READ/6000
- BookManager READ/DOS
- BookManager READ/MVS
- BookManager READ/VM
- BookManager READ for Windows

PostScript format

The MQSeries library is provided in PostScript (.PS) format with many MQSeries Version 2 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

Windows Help format

The *MQSeries for Windows User's Guide* is provided in Windows Help format with MQSeries for Windows, Version 2.0 and MQSeries for Windows, Version 2.1.

MQSeries information available on the Internet

The MQSeries product family Web site is at:

<http://www.ibm.com/software/mqseries/>

By following links from this Web site you can:

- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML and PDF formats.
- Download an MQSeries SupportPac[™].

MQSeries on the Internet

Index

A

- Accept Direct Requests attribute 480
- Accept Truncated Message attribute 478
- addElement
 - AmMessage (C++) 191
 - AmMessage (Java) 389
- addFilter
 - AmMessage (C++) 191
 - AmMessage (Java) 390
- addTopic
 - AmMessage (C++) 191
 - AmMessage (Java) 390
- administration tool 457
- AIX
 - C++ applications 161
 - C applications 30
 - installation 428
 - Java applications 366
 - prerequisites 425
- AMB constants 509
- amBackout 40
- amBegin 41
- amBrowseMsg 42
- AMBRW constants 509
- AmBytes (C++)
 - cmp 213
 - constructors 213
 - cpy 214
 - dataPtr 214
 - destructor 214
 - length 214
 - operators 214
 - overview 179
 - pad 214
- AMCC constants 509
- amCommit 44
- AmConstants (Java) 411
- AMDEF constants 509
- AmDistributionList (C++)
 - clearErrorCodes 205
 - close 205
 - enableWarnings 205
 - getLastErrorStatus 205
 - getName 205
 - getSender 205
 - getSenderCount 205
 - open 205
 - overview 175
 - send 206
 - sendFile 206
- AmDistributionList (Java)
 - clearErrorCodes 403
 - close 403
 - enableWarnings 403
 - getLastErrorStatus 403
 - getName 403
 - getSender 403
 - getSenderCount 403
 - open 403
 - overview 376
 - send 404
- AmDistributionList (Java) (*continued*)
 - sendFile 404
- AMDLCCL 332
- AMDLCLEC 332
- AMDLTGTL 332
- AMDLTGNA 333
- AMDLTGSH 334
- AMDLOP 334
- AMDLSN 335
- AMDLSNFL 335
- amDstClearErrorCodes 126
- amDstClose 126
- amDstGetLastError 127
- amDstGetName 127
- amDstGetSenderCount 128
- amDstGetSenderHandle 128
- amDstOpen 128
- amDstSend 129
- amDstSendFile 130
- AMDT constants 509
- AMELEM structure 24, 237
- AmElement (C++)
 - constructor 215
 - getName 215
 - getValue 215
 - getVersion 215
 - overview 179
 - setVersion 215
 - toString 215
 - using 157
- AmElement (Java)
 - constructor 412
 - getName 412
 - getValue 412
 - getVersion 412
 - overview 380
 - setVersion 412
 - toString 412
 - using 362
- AMENC constants 510
- AmErrorException (C++)
 - getClassName 221
 - getCompletionCode 221
 - getMethodName 221
 - getReasonCode 221
 - getSource 221
 - overview 181
 - toString 221
- AmErrorException (Java)
 - getClassName 416
 - getCompletionCode 416
 - getMethodName 416
 - getReasonCode 416
 - getSource 416
 - overview 381
 - toString 416
- AmException (C++)
 - getClassName 220
 - getCompletionCode 220
 - getMethodName 220
- AmException (C++) (*continued*)
 - getReasonCode 220
 - getSource 220
 - overview 181
 - toString 220
 - using 157
- AmException (Java)
 - getClassName 415
 - getCompletionCode 415
 - getMethodName 415
 - getReasonCode 415
 - getSource 415
 - overview 381
 - toString 415
 - using 363
- AMFB constants 510
- AMFMT constants 510
- AMGF constants 510
- AMGRP constants 510
- AMH constants 510
- AMHBACK 247
- AMHBEGIN 248
- AMHBRMS 249
- AMHCMIT 251
- AMHINIT 252
- AMHPB 253
- AMHRCFL 254
- AMHRCMS 256
- AMHRCPB 258
- AMHRCRQ 260
- AMHSB 266
- AMHSNFL 262
- AMHSNMS 263
- AMHSNRQ 264
- AMHSNRS 265
- AMHTERM 267
- AMHUN 268
- amInitialize 45
- AMLEN constants 510
- AMMCD constants 510
- AmMessage (C++)
 - addElement 191
 - addFilter 191
 - addTopic 191
 - clearErrorCodes 191
 - deleteElement 191
 - deleteFilter 191
 - deleteNamedElement 192
 - deleteTopic 192
 - enableWarnings 192
 - getCCSID 192
 - getCorrelationId 192
 - getDataLength 192
 - getDataOffset 192
 - getElement 193
 - getElementCCSID 193
 - getElementCount 193
 - getEncoding 193
 - getFilter 193
 - getFilterCount 193
 - getFormat 193

AmMessage (C++) (continued)

- getGroupStatus 194
- getLastErrorStatus 194
- getMessageId 194
- getName 194
- getNamedElement 194
- getNamedElementCount 194
- getReportCode 195
- getTopic 195
- getTopicCount 195
- getType 195
- overview 171
- readBytes 195
- reset 195
- setCCSID 196
- setCorrelationId 196
- setDataOffset 196
- setElementCCSID 196
- setEncoding 196
- setFormat 197
- setGroupStatus 197
- writeBytes 197

AmMessage (Java)

- addElement 389
- addFilter 390
- addTopic 390
- clearErrorCodes 390
- deleteElement 390
- deleteFilter 390
- deleteNamedElement 390
- deleteTopic 391
- enableWarnings 391
- getCCSID 391
- getCorrelationId 391
- getDataLength 391
- getDataOffset 391
- getElement 391
- getElementCount 391
- getEncoding 392
- getFilter 392
- getFilterCount 392
- getFormat 392
- getGroupStatus 392
- getLastErrorStatus 392
- getMessageId 393
- getName 393
- getNamedElement 393
- getNamedElementCount 393
- getReportCode 393
- getTopic 393
- getTopicCount 393
- getType 394
- overview 372
- readBytes 394
- reset 394
- setCCSID 394
- setCorrelationId 394
- setDataOffset 394
- setEncoding 395
- setFormat 395
- setGroupStatus 395
- writeBytes 395

AMMSADEL 298

AMMSADFI 298

AMMSADTO 299

AMMSCLEC 299

AMMSDEEL 299

AMMSDEFI 300

AMMSDENE 300

AMMSDETO 301

amMsgAddElement 91

amMsgAddFilter 91

amMsgAddStreamName 108

amMsgAddTopic 92

amMsgClearErrorCodes 92

amMsgDeleteElement 92

amMsgDeleteFilter 93

amMsgDeleteNamedElement 93

amMsgDeleteTopic 94

AMMSGELC 301

amMsgGetCCSID 94

amMsgGetCorrelId 94

amMsgGetDataLength 95

amMsgGetDataOffset 95

amMsgGetElement 95

amMsgGetElementCCSID 96

amMsgGetElementCount 96

amMsgGetEncoding 96

amMsgGetFilter 97

amMsgGetFilterCount 97

amMsgGetFormat 98

amMsgGetGroupStatus 98

amMsgGetLastError 99

amMsgGetMsgId 99

amMsgGetName 100

amMsgGetNamedElement 100

amMsgGetNamedElementCount 101

amMsgGetPubTimeStamp 108

amMsgGetReportCode 101

amMsgGetStreamName 109

amMsgGetTopic 102

amMsgGetTopicCount 103

amMsgGetType 102

amMsgReadBytes 103

amMsgReset 103

amMsgSetCCSID 104

amMsgSetCorrelId 104

amMsgSetDataOffset 104

amMsgSetElementCCSID 105

amMsgSetEncoding 105

amMsgSetFormat 106

amMsgSetGroupStatus 106

AMMSGTCC 301

AMMSGTCI 302

AMMSGTDL 302

AMMSGTDO 302

AMMSGTEC 303

AMMSGTEL 303

AMMSGTEN 304

AMMSGTFC 304

AMMSGTFI 305

AMMSGTFO 305

AMMSGTGS 306

AMMSGTLE 306

AMMSGTMI 307

AMMSGTNA 307

AMMSGTNC 308

AMMSGTNE 308

AMMSGTRC 309

AMMSGTTC 310

AMMSGTTO 309

AMMSGTTY 310

amMsgWriteBytes 107

AMMSREBY 311

AMMSRS 311

AMMSSELC 313

AMMSSTCC 312

AMMSSTCI 312

AMMSSTDO 312

AMMSSTEN 313

AMMSSTFO 314

AMMSSTGS 314

AMMSWRBY 315

AMMT constants 510

AmObject (C++)

- clearErrorCodes 216
- getLastErrorStatus 216
- getName 216
- overview 179

AmObject (Java)

- clearErrorCodes 413
- getLastErrorStatus 413
- getName 413
- overview 380

AMPBCL 337

AMPBCLC 337

AMPBGTC 337

AMPBGTEN 338

AMPBGTLE 338

AMPBGTNA 339

AMPBOP 339

AMPBPB 340

AMPOCLEC 348

AMPOGTLE 348

AMPOGTNA 349

AMPOGTWT 349

amPolClearErrorCodes 142

amPolGetLastError 142

amPolGetName 143

amPolGetWaitTime 143

AmPolicy (C++)

- clearErrorCodes 212
- enableWarnings 212
- getLastErrorStatus 212
- getName 212
- getWaitTime 212
- overview 178
- setWaitTime 212

AmPolicy (Java)

- clearErrorCodes 410
- enableWarnings 410
- getLastErrorStatus 410
- getName 410
- getWaitTime 410
- overview 379
- setWaitTime 410

amPolSetWaitTime 143

AMPOSTWT 350

AMPS constants 511

amPubClearErrorCodes 131

amPubClose 131

amPubGetCCSID 131

amPubGetEncoding 132

amPubGetLastError 132

amPubGetName 133

amPublish 46

AmPublisher (C++)

- clearErrorCodes 207
- close 207
- enableWarnings 207
- getCCSID 207

AmPublisher (C++) (continued)
 getEncoding 207
 getLastErrorStatus 207
 getName 207
 open 208
 overview 176
 publish 208

AmPublisher (Java)
 clearErrorCodes 405
 close 405
 enableWarnings 405
 getCCSID 405
 getEncoding 405
 getLastErrorStatus 405
 getName 405
 open 406
 overview 377
 publish 406

amPubOpen 133
 amPubPublish 134

AMRC constants 512
 AMRCBR 322
 AMRCBRSE 324
 AMRCCL 326
 AMRCCLEC 325
 AMRCGTD 326
 AMRCGTLE 327
 AMRCGTNA 327
 AMRCGTQN 328
 AMRCOP 328
 AMRCRC 329
 AMRCRCFL 330
 AMRCSTQN 331
 amRcvBrowse 115
 amRcvBrowseSelect 117
 amRcvClearErrorCodes 118
 amRcvClose 119
 amRcvGetDefnType 119
 amRcvGetLastError 120
 amRcvGetName 120
 amRcvGetQueueName 121
 amRcvOpen 121
 amRcvReceive 122
 amRcvReceiveFile 124
 amRcvSetQueueName 125
 amReceiveFile 47
 amReceiveMsg 49
 amReceivePublication 51

AmReceiver (C++)
 browse 201
 clearErrorCodes 202
 close 202
 enableWarnings 202
 getDefinitionType 202
 getLastErrorStatus 203
 getName 203
 getQueueName 203
 open 203
 overview 174
 receive 203
 receiveFile 204, 402
 setQueueName 204

AmReceiver (Java)
 browse 399
 clearErrorCodes 400
 close 400
 enableWarnings 400

AmReceiver (Java) (continued)
 getDefinitionType 400
 getLastErrorStatus 401
 getName 401
 getQueueName 401
 open 401
 overview 375
 receive 401
 setQueueName 402

amReceiveRequest 53

AMSBCL 341
 AMSBCLC 341
 AMSBGTCC 342
 AMSBGTD 342
 AMSBGTEN 343
 AMSBGTLE 343
 AMSBGTNA 344
 AMSBGTQN 344
 AMSBOP 345
 AMSBRC 345
 AMSBSB 346
 AMSBSTQN 346
 AMSBUN 347

AMSD constants 515
 AMSEBG 284
 AMSECL 285
 AMSECLEC 284
 AMSECM 285
 AMSECR 286
 AMSECRDL 286
 AMSECRMS 287
 AMSECRPB 288
 AMSECRPO 287
 AMSECRRC 288
 AMSECRSB 289
 AMSECRSN 289
 AMSEDL 290
 AMSEDLDL 290
 AMSEDLMS 290
 AMSEDLPB 291
 AMSEDLPO 291
 AMSEDLRC 291
 AMSEDLRSB 292
 AMSEDLN 292
 AMSEGLDL 292
 AMSEGLHMS 293
 AMSEGLHPB 294
 AMSEGLHPO 294
 AMSEGLHRC 294
 AMSEGLHSB 295
 AMSEGLHSN 295
 AMSEGLTLE 293

AmSender (C++)
 clearErrorCodes 198
 close 198
 enableWarnings 198
 getCCSID 198
 getEncoding 198
 getLastErrorStatus 199
 getName 199
 open 199
 overview 173
 send 199
 sendFile 200

AmSender (Java)
 clearErrorCodes 396
 close 396

AmSender (Java) (continued)
 enableWarnings 396
 getCCSID 396
 getEncoding 396
 getLastErrorStatus 397
 getName 397
 open 397
 overview 374
 send 397
 sendFile 398

amSendFile 55
 amSendMsg 56
 amSendRequest 57
 amSendResponse 58

AMSEOP 296
 AMSERB 296
 amSesBegin 78
 amSesClearErrorCodes 78
 amSesClose 79
 amSesCommit 79
 amSesCreate 79
 amSesCreateDistList 80
 amSesCreateMessage 80
 amSesCreatePolicy 80
 amSesCreatePublisher 81
 amSesCreateReceiver 81
 amSesCreateSender 82
 amSesCreateSubscriber 82
 amSesDelete 83
 amSesDeleteDistList 83
 amSesDeleteMessage 83
 amSesDeletePolicy 84
 amSesDeletePublisher 84
 amSesDeleteReceiver 84
 amSesDeleteSender 85
 amSesDeleteSubscriber 85
 amSesGetDistListHandle 85
 amSesGetLastError 86
 amSesGetMessageHandle 86
 amSesGetPolicyHandle 87
 amSesGetPublisherHandle 87
 amSesGetReceiverHandle 87
 amSesGetSenderHandle 88
 amSesGetSubscriberHandle 88
 amSesOpen 88
 amSesRollback 89

AmSession (C++)
 begin 186
 clearErrorCodes 186
 close 186
 commit 186
 createDistributionList 186
 createMessage 187
 createPolicy 187
 createPublisher 187
 createReceiver 187
 createSender 187
 createSubscriber 187
 deleteDistributionList 188
 deleteMessage 188
 deletePolicy 188
 deletePublisher 188
 deleteReceiver 188
 deleteSender 188
 deleteSubscriber 188
 enableWarnings 188
 getLastErrorStatus 189

- AmSession (C++) (*continued*)
 - getName 189
 - getTraceLevel 189
 - getTraceLocation 189
 - open 189
 - overview 169
 - rollback 189
 - transaction coordination 186
 - unit of work 186
- AmSession (Java)
 - begin 386
 - clearErrorCodes 386
 - close 386
 - commit 386
 - createDistributionList 386
 - createMessage 387
 - createPolicy 387
 - createPublisher 387
 - createReceiver 387
 - createSender 387
 - createSubscriber 387
 - enableWarnings 388
 - getLastErrorStatus 388
 - getName 388
 - getTraceLevel 388
 - getTraceLocation 388
 - open 388
 - overview 371
 - rollback 388
 - transaction coordination 386
 - unit of work 386
- AmSessionFactory (C++)
 - constructors 184
 - createSession 184
 - deleteSession 184
 - getFactoryName 184
 - getLocalHost 184
 - getRepository 184
 - getTraceLevel 184
 - getTraceLocation 184
 - overview 168
 - setLocalHost 185
 - setRepository 185
 - setTraceLevel 185
 - setTraceLocation 185
- AmSessionFactory (Java)
 - constructor 384
 - createSession 384
 - getFactoryName 384
 - getLocalHost 384
 - getRepository 384
 - getTraceLevel 384
 - getTraceLocation 384
 - overview 370
 - setLocalHost 384
 - setRepository 385
 - setTraceLevel 385
 - setTraceLocation 385
- AMSNCL 317
- AMSNCLEC 316
- amSndClearErrorCodes 110
- amSndClose 110
- amSndGetCCSID 111
- amSndGetEncoding 111
- amSndGetLastError 111
- amSndGetName 112
- amSndOpen 112
- amSndSend 113
- amSndSendFile 114
- AMNGTCC 317
- AMNGTEN 317
- AMNGTLE 318
- AMNGTNA 318
- AMSNOP 319
- AMSNSN 319
- AMSNSNFL 320
- AmStatus (C++)
 - constructor 217
 - getCompletionCode 217
 - getReasonCode 217
 - getReasonCode2 217
 - overview 179
 - toString 217
 - using 157
- AmStatus (Java)
 - constructor 414
 - getCompletionCode 414
 - getReasonCode 414
 - getReasonCode2 414
 - overview 380
 - toString 414
 - using 363
- AmString (C++)
 - cat 218
 - cmp 218
 - constructors 218
 - contains 218
 - cpy 218
 - destructor 218
 - length 219
 - operators 219
 - overview 180
 - pad 219
 - split 219
 - strip 219
 - text 219
 - truncate 219
- amSubClearErrorCodes 135
- amSubClose 135
- amSubGetCCSID 135
- amSubGetDefnType 136
- amSubGetEncoding 136
- amSubGetLastError 137
- amSubGetName 137
- amSubGetQueueName 138
- amSubOpen 138
- amSubReceive 139
- amSubscribe 59
- AmSubscriber (C++)
 - clearErrorCodes 209
 - close 209
 - enableWarnings 209
 - getCCSID 209
 - getDefinitionType 209
 - getEncoding 209
 - getLastErrorStatus 209
 - getName 210
 - getQueueName 210
 - open 210
 - overview 177
 - receive 210
 - setQueueName 210
 - subscribe 211
 - unsubscribe 211
- AmSubscriber (Java)
 - clearErrorCodes 407
 - close 407
 - enableWarnings 407
 - getCCSID 407
 - getDefinitionType 407
 - getEncoding 407
 - getLastErrorStatus 407
 - getName 408
 - getQueueName 408
 - open 408
 - overview 378
 - receive 408
 - setQueueName 408
 - subscribe 409
 - unsubscribe 409
- amSubSetQueueName 139
- amSubSubscribe 140
- amSubUnsubscribe 141
- amtc.h header 29
- AMTELEMV and AMTELEML
 - copybooks 237
- amTerminate 60
- amUnsubscribe 61
- AmWarningException (C++)
 - getClassName 222
 - getCompletionCode 222
 - getMethodName 222
 - getReasonCode 222
 - getSource 222
 - overview 181
 - toString 222
- AmWarningException (Java)
 - getClassName 417
 - getCompletionCode 417
 - getMethodName 417
 - getReasonCode 417
 - getSource 417
 - overview 381
 - toString 417
- AMWT constants 515
- Anonymous Registration attribute 479, 480
- appearance of text in this book xx
- Application Group attribute 476
- application messaging interface
 - basic model 7
 - description 4
 - interoperability 3
 - introduction 3
 - main components 3
 - programming languages 4
 - receiving messages 3
 - sending messages 3
- applications, building
 - C 29
 - C++ 161
 - COBOL 241
 - Java 366
- applications, writing
 - C 16
 - C++ 151
 - COBOL 230
 - Java 357
- AS/400
 - C++ applications 162
 - C applications 31

AS/400 (*continued*)
 installation 432
 Java applications 366
 prerequisites 425
 sample programs 461
 attributes, policy
 general 476
 initialization 475
 publish 480
 receive 478
 send 476
 subscribe 479
 Available Service Points attribute 474

B

base classes
 C++ 149, 183
 Java 355, 383
 begin
 AmSession (C++) 186
 AmSession (Java) 386
 bibliography 521
 Bind On Open attribute 476
 BookManager 523
 Boolean constants 509
 Broker Service attribute 474
 browse
 AmReceiver (C++) 201
 AmReceiver (Java) 399
 constants 509
 building applications
 C 29
 C++ 161
 COBOL 241
 Java 366

C

C++ applications
 AIX 161
 AS/400 162
 HP-UX 163
 Solaris 164
 Windows 165
 C++ interface
 overview 167
 reference information 183
 using 149
 C applications
 AIX 30
 AS/400 31
 HP-UX 32
 OS/390 33
 Solaris 34
 Windows 35
 C high-level interface
 equivalent object interface
 functions 74
 overview 38
 reference information 39
 using 13
 C object interface
 overview 63
 reference information 77
 using 13

cache, repository and local host
 (OS/390) 454
 CCSID attribute 472
 class (C++)
 base 149, 183
 exception 150, 183
 helper 150, 183
 class (Java)
 base 355, 383
 exception 356, 383
 helper 356, 383
 clearErrorCodes
 AmDistributionList (C++) 205
 AmDistributionList (Java) 403
 AmMessage (C++) 191
 AmMessage (Java) 390
 AmObject (C++) 216
 AmObject (Java) 413
 AmPolicy (C++) 212
 AmPolicy (Java) 410
 AmPublisher (C++) 207
 AmPublisher (Java) 405
 AmReceiver (C++) 202
 AmReceiver (Java) 400
 AmSender (C++) 198
 AmSender (Java) 396
 AmSession (C++) 186
 AmSession (Java) 386
 AmSubscriber (C++) 209
 AmSubscriber (Java) 407
 close
 AmDistributionList (C++) 205
 AmDistributionList (Java) 403
 AmPublisher (C++) 207
 AmPublisher (Java) 405
 AmReceiver (C++) 202
 AmReceiver (Java) 400
 AmSender (C++) 198
 AmSender (Java) 396
 AmSession (C++) 186
 AmSession (Java) 386
 AmSubscriber (C++) 209
 AmSubscriber (Java) 407
 closing a session
 C 16
 C++ 151
 COBOL 230
 Java 357
 COBOL applications
 OS/390 241
 COBOL high-level interface
 equivalent object interface
 functions 280
 overview 244
 reference information 246
 using 227
 COBOL object interface
 overview 269
 reference information 283
 using 227
 commit
 AmSession (C++) 186
 AmSession (Java) 386
 compilers 426
 completion code constants 509
 connecting to MQSeries 458
 Connection Mode attribute 475

Connection Name attribute 475
 Connection Type attribute 475
 constants
 Boolean 509
 browse 509
 completion codes 509
 definition type 509
 encoding 510
 feedback codes 510
 format 510
 group status 510
 handle 510
 Message Content Descriptor tag
 names 510
 message types 510
 publish/subscribe 511
 reason codes 512
 service and policy definitions 509
 string length 510
 system default names and
 handles 515
 wait time 515
 content-based publish/subscribe 459
 Convert attribute 478
 createDistributionList
 AmSession (C++) 186
 AmSession (Java) 386
 createMessage
 AmSession (C++) 187
 AmSession (Java) 387
 createPolicy
 AmSession (C++) 187
 AmSession (Java) 387
 createPublisher
 AmSession (C++) 187
 AmSession (Java) 387
 createReceiver
 AmSession (C++) 187
 AmSession (Java) 387
 createSender
 AmSession (C++) 187
 AmSession (Java) 387
 createSession
 AmSessionFactory (C++) 184
 AmSessionFactory (Java) 384
 createSubscriber
 AmSession (C++) 187
 AmSession (Java) 387
 creating MQSeries objects 460
 creating objects
 C++ 151
 Java 357

D

data types, C 29
 datagram
 C 16
 C++ 152
 COBOL 230
 Java 357
 Default Format attribute 472
 Default MCD Domain attribute 472
 Default MCD Format attribute 472
 Default MCD Set attribute 472
 Default MCD Type attribute 472

- default objects
 - C 14
 - C++ 150
 - COBOL 228
 - Java 356
 - system 470
 - definition
 - distribution list 474
 - policy 469, 475
 - publisher 474
 - service 469
 - service point (sender/receiver) 472
 - subscriber 474
 - system provided 470
 - Definition Type attribute 472
 - definition type constants 509
 - Delete On Close attribute 478
 - deleteDistributionList
 - AmSession (C++) 188
 - deleteElement
 - AmMessage (C++) 191
 - AmMessage (Java) 390
 - deleteFilter
 - AmMessage (C++) 191
 - AmMessage (Java) 390
 - deleteMessage
 - AmSession (C++) 188
 - deleteNamedElement
 - AmMessage (C++) 192
 - AmMessage (Java) 390
 - deletePolicy
 - AmSession (C++) 188
 - deletePublisher
 - AmSession (C++) 188
 - deleteReceiver
 - AmSession (C++) 188
 - deleteSender
 - AmSession (C++) 188
 - deleteSession
 - AmSessionFactory (C++) 184
 - deleteSubscriber
 - AmSession (C++) 188
 - deleteTopic
 - AmMessage (C++) 192
 - AmMessage (Java) 391
 - deleting C++ objects 152
 - directory structure
 - AIX 430
 - AS/400 433
 - HP-UX 438
 - OS/390 441
 - Solaris 445
 - Windows 448
 - disk space 425
 - distribution list definition 474
 - distribution list interface
 - overview (C) 70
 - overview (C++) 175
 - overview (COBOL) 276
 - overview (Java) 376
 - distribution list interface (C)
 - amDstClearErrorCodes 126
 - amDstClose 126
 - amDstGetLastError 127
 - amDstGetName 127
 - amDstGetSenderCount 128
 - amDstGetSenderHandle 128
 - distribution list interface (C) (*continued*)
 - amDstOpen 128
 - amDstSend 129
 - amDstSendFile 130
 - distribution list interface (COBOL)
 - AMDLCCL 332
 - AMDLCLEC 332
 - AMDLGTL 332
 - AMDLTGNA 333
 - AMDLTGSC 333
 - AMDLTGSH 334
 - AMDLOP 334
 - AMDLSN 335
 - AMDLSNFL 335
 - Dynamic Queue Prefix attribute 472
- E**
- elements, name/value
 - C 24
 - C++ 157
 - COBOL 237
 - Java 362
 - enableWarnings
 - AmDistributionList (C++) 205
 - AmDistributionList (Java) 403
 - AmMessage (C++) 192
 - AmMessage (Java) 391
 - AmPolicy (C++) 212
 - AmPolicy (Java) 410
 - AmPublisher (C++) 207
 - AmPublisher (Java) 405
 - AmReceiver (C++) 202
 - AmReceiver (Java) 400
 - AmSender (C++) 198
 - AmSender (Java) 396
 - AmSession (C++) 188
 - AmSession (Java) 388
 - AmSubscriber (C++) 209
 - AmSubscriber (Java) 407
 - Encoding attribute 472
 - encoding constants 510
 - error handling
 - C 25
 - C++ 157
 - COBOL 239
 - Java 363
 - examples
 - C 16
 - C++ 151
 - COBOL 230
 - Java 357
 - Exception Action attribute 476
 - exception classes
 - C++ 150, 183
 - Java 356, 383
 - Expiry Interval attribute 476
- F**
- failure (of AMI program)
 - common causes 493
 - reason codes 492
 - symptom report (OS/390) 492
 - symptom report (UNIX and Windows) 492
 - feedback codes 510
 - Field Disposition attribute 478
 - field limits
 - C 28
 - C++ 160
 - COBOL 240
 - Java 365
 - file transfer
 - C 21
 - C++ 155
 - COBOL 235
 - Java 361
 - filters 459
 - filters for publish/subscribe 459
 - format constants 510
- G**
- getCCSID
 - AmMessage (C++) 192
 - AmMessage (Java) 391
 - AmPublisher (C++) 207
 - AmPublisher (Java) 405
 - AmSender (C++) 198
 - AmSender (Java) 396
 - AmSubscriber (C++) 209
 - AmSubscriber (Java) 407
 - getClassname
 - AmErrorException (C++) 221
 - AmErrorException (Java) 416
 - AmException (C++) 220
 - AmException (Java) 415
 - AmWarningException (C++) 222
 - AmWarningException (Java) 417
 - getCompletionCode
 - AmErrorException (C++) 221
 - AmErrorException (Java) 416
 - AmException (C++) 220
 - AmException (Java) 415
 - AmStatus (C++) 217
 - AmStatus (Java) 414
 - AmWarningException (C++) 222
 - AmWarningException (Java) 417
 - getCorrelationId
 - AmMessage (C++) 192
 - AmMessage (Java) 391
 - getDataLength
 - AmMessage (C++) 192
 - AmMessage (Java) 391
 - getDataOffset
 - AmMessage (C++) 192
 - AmMessage (Java) 391
 - getDefinitionType
 - AmReceiver (C++) 202
 - AmReceiver (Java) 400
 - AmSubscriber (C++) 209
 - AmSubscriber (Java) 407
 - getElement
 - AmMessage (C++) 193
 - AmMessage (Java) 391
 - getElementCCSID
 - AmMessage (C++) 193
 - getElementCount
 - AmMessage (C++) 193
 - AmMessage (Java) 391
 - getEncoding
 - AmMessage (C++) 193
 - AmMessage (Java) 392
 - AmPublisher (C++) 207

getEncoding (*continued*)
 AmPublisher (Java) 405
 AmSender (C++) 198
 AmSender (Java) 396
 AmSubscriber (C++) 209
 AmSubscriber (Java) 407
 getFactoryName
 AmSessionFactory (C++) 184
 AmSessionFactory (Java) 384
 getFilter
 AmMessage (C++) 193
 AmMessage (Java) 392
 getFilterCount
 AmMessage (C++) 193
 AmMessage (Java) 392
 getFormat
 AmMessage (C++) 193
 AmMessage (Java) 392
 getGroupStatus
 AmMessage (C++) 194
 AmMessage (Java) 392
 getLastErrorStatus
 AmDistributionList (C++) 205
 AmDistributionList (Java) 403
 AmMessage (C++) 194
 AmMessage (Java) 392
 AmObject (C++) 216
 AmObject (Java) 413
 AmPolicy (C++) 212
 AmPolicy (Java) 410
 AmPublisher (C++) 207
 AmPublisher (Java) 405
 AmReceiver (C++) 203
 AmReceiver (Java) 401
 AmSender (C++) 199
 AmSender (Java) 397
 AmSession (C++) 189
 AmSession (Java) 388
 AmSubscriber (C++) 209
 AmSubscriber (Java) 407
 getLocalHost
 AmSessionFactory (C++) 184
 AmSessionFactory (Java) 384
 getMessageId
 AmMessage (C++) 194
 AmMessage (Java) 393
 getMethodName
 AmErrorException (C++) 221
 AmErrorException (Java) 416
 AmException (C++) 220
 AmException (Java) 415
 AmWarningException (C++) 222
 AmWarningException (Java) 417
 getName
 AmDistributionList (C++) 205
 AmDistributionList (Java) 403
 AmElement (C++) 215
 AmElement (Java) 412
 AmMessage (C++) 194
 AmMessage (Java) 393
 AmObject (C++) 216
 AmObject (Java) 413
 AmPolicy (C++) 212
 AmPolicy (Java) 410
 AmPublisher (C++) 207
 AmPublisher (Java) 405
 AmReceiver (C++) 203
 AmReceiver (Java) 401
 AmSender (C++) 199
 AmSender (Java) 397
 AmSession (C++) 189
 AmSession (Java) 388
 AmSubscriber (C++) 210
 AmSubscriber (Java) 408
 getNamedElement
 AmMessage (C++) 194
 AmMessage (Java) 393
 getNamedElementCount
 AmMessage (C++) 194
 AmMessage (Java) 393
 getQueueName
 AmReceiver (C++) 203
 AmReceiver (Java) 401
 AmSubscriber (C++) 210
 AmSubscriber (Java) 408
 getReasonCode
 AmErrorException (C++) 221
 AmErrorException (Java) 416
 AmException (C++) 220
 AmException (Java) 415
 AmStatus (C++) 217
 AmStatus (Java) 414
 AmWarningException (C++) 222
 AmWarningException (Java) 417
 getReasonCode2
 AmStatus (C++) 217
 AmStatus (Java) 414
 getReportCode
 AmMessage (C++) 195
 AmMessage (Java) 393
 getRepository
 AmSessionFactory (C++) 184
 AmSessionFactory (Java) 384
 getSender
 AmDistributionList (C++) 205
 AmDistributionList (Java) 403
 getSenderCount
 AmDistributionList (C++) 205
 AmDistributionList (Java) 403
 getSource
 AmErrorException (C++) 221
 AmErrorException (Java) 416
 AmException (C++) 220
 AmException (Java) 415
 AmWarningException (C++) 222
 AmWarningException (Java) 417
 getTopic
 AmMessage (C++) 195
 AmMessage (Java) 393
 getTopicCount
 AmMessage (C++) 195
 AmMessage (Java) 393
 getTraceLevel
 AmSession (C++) 189
 AmSession (Java) 388
 AmSessionFactory (C++) 184
 AmSessionFactory (Java) 384
 getTraceLocation
 AmSession (C++) 189
 AmSession (Java) 388
 AmSessionFactory (C++) 184
 AmSessionFactory (Java) 384
 getType
 AmMessage (C++) 195
 AmMessage (Java) 394
 getValue
 AmElement (C++) 215
 AmElement (Java) 412
 getVersion
 AmElement (C++) 215
 AmElement (Java) 412
 getWaitTime
 AmPolicy (C++) 212
 AmPolicy (Java) 410
 glossary 519
 group status constants 510

H

handle constants 510
 Handle Poison Message attribute 478
 header file
 C 29
 C++ 161
 helper classes
 C++ 150, 183
 Java 356, 383
 helper macros 108
 high-level interface
 equivalent object interface
 functions 74
 using 13
 high-level interface (C)
 amBackout 40
 amBegin 41
 amBrowseMsg 42
 amCommit 44
 amInitialize 45
 amPublish 46
 amReceiveFile 47
 amReceiveMsg 49
 amReceivePublication 51
 amReceiveRequest 53
 amSendFile 55
 amSendMsg 56
 amSendRequest 57
 amSendResponse 58
 amSubscribe 59
 amTerminate 60
 amUnsubscribe 61
 overview 38
 reference information 39
 high-level interface (COBOL)
 AMHBACK 247
 AMHBEGIN 248
 AMHBRMS 249
 AMHCMIT 251
 AMHINIT 252
 AMHPB 253
 AMHRCFL 254
 AMHRCMS 256
 AMHRCPB 258
 AMHRCRQ 260
 AMHSB 266
 AMHSNFL 262
 AMHSNMS 263
 AMHSNRQ 264
 AMHSNRS 265
 AMHTERM 267
 AMHUN 268

high-level interface (COBOL) *(continued)*

- equivalent object interface
- functions 280
- overview 244
- reference information 246
- using 227

HP-UX

- C++ applications 163
- C applications 32
- installation 436
- Java applications 366
- prerequisites 425

HTML (Hypertext Markup Language) 522

Hypertext Markup Language (HTML) 522

I

Implicit Open attribute 476, 478

include file

- C 29
- C++ 161

Inform If Retained attribute 479

initial values for structures 29

installation

- administration tool 457
- AIX 428
- AS/400 432
- HP-UX 436
- OS/390 440
- prerequisites 425
- Solaris 443
- Windows 447

interface

- C++ 167, 183
- C high-level 37, 38
- C object 63, 77
- COBOL high-level 243, 244
- COBOL object 269, 283
- Java 369, 383

interoperability 3

J

jar file (Java) 366

Java applications

- AIX 366
- AS/400 366
- HP-UX 366
- Solaris 366
- Windows 366

Java interface

- overview 369
- reference information 383
- using 355

L

Leave Queue Open attribute 476, 478

local host cache (OS/390) 454

local host file 450

local host file (OS/390) 453

M

macros, helper 108

Message Content Descriptor tag names 510

Message Context attribute 476

message interface

- overview (C) 66
- overview (C++) 171
- overview (COBOL) 272
- overview (Java) 372

message interface (C)

- amMsgAddElement 91
- amMsgAddFilter 91
- AmMsgAddStreamName 108
- amMsgAddTopic 92
- amMsgClearErrorCodes 92
- amMsgDeleteElement 92
- amMsgDeleteFilter 93
- amMsgDeleteNamedElement 93
- amMsgDeleteTopic 94
- amMsgGetCCSID 94
- amMsgGetCorrelId 94
- amMsgGetDataLength 95
- amMsgGetDataOffset 95
- amMsgGetElement 95
- amMsgGetElementCCSID 96
- amMsgGetElementCount 96
- amMsgGetEncoding 96
- amMsgGetFilter 97
- amMsgGetFilterCount 97
- amMsgGetFormat 98
- amMsgGetGroupStatus 98
- amMsgGetLastError 99
- amMsgGetMsgId 99
- amMsgGetName 100
- amMsgGetNamedElement 100
- amMsgGetNamedElementCount 101
- AmMsgGetPubTimeStamp 108
- amMsgGetReportCode 101
- AmMsgGetStreamName 109
- amMsgGetTopic 102
- amMsgGetTopicCount 103
- amMsgGetType 102
- amMsgReadBytes 103
- amMsgReset 103
- amMsgSetCCSID 104
- amMsgSetCorrelId 104
- amMsgSetDataOffset 104
- amMsgSetElementCCSID 105
- amMsgSetEncoding 105
- amMsgSetFormat 106
- amMsgSetGroupStatus 106
- amMsgWriteBytes 107

message interface (COBOL)

- AMMSADEL 298
- AMMSADFI 298
- AMMSADTO 299
- AMMSCLEC 299
- AMMSDEEL 299
- AMMSDEFI 300
- AMMSDENE 300
- AMMSDETO 301
- AMMSGELC 301
- AMMSGTCC 301
- AMMSGTCI 302
- AMMSGTDL 302

message interface (COBOL) *(continued)*

- AMMSGTDO 302
- AMMSGTEC 303
- AMMSGTEL 303
- AMMSGTEN 304
- AMMSGTFC 304
- AMMSGTFI 305
- AMMSGTFO 305
- AMMSGTGS 306
- AMMSGTLE 306
- AMMSGTMI 307
- AMMSGTNA 307
- AMMSGTNC 308
- AMMSGTNE 308
- AMMSGTRC 309
- AMMSGTTC 310
- AMMSGTTO 309
- AMMSGTTY 310
- AMMSREBY 311
- AMMSRS 311
- AMMSSELC 313
- AMMSSTCC 312
- AMMSSTCI 312
- AMMSSTDO 312
- AMMSSTEN 313
- AMMSSTFO 314
- AMMSSTGS 314
- AMMSWRBY 315

message types 510

messages 4

messages, poison 478

messages, publish/subscribe

- C 22
- C++ 156

COBOL 235

Java 361

messages, receiving

- C 18
- C++ 153
- COBOL 232
- Java 359

messages, request/response

- C 19
- C++ 154
- COBOL 234
- Java 360

messages, sending

- C 16
- C++ 152
- COBOL 230
- Java 357

model of the AMI 7

Model Queue Name attribute 472

MQSeries client

- connecting to 458
- prerequisites 426

MQSeries environment 426

MQSeries function calls

- C 27
- C++ 159
- COBOL 240
- Java 365

MQSeries Integrator V2 472

- Migrating API applications to 459
- Using the AMI with 458

MQSeries Integrator Version 1, using 458

- MQSeries objects, creating 460
- MQSeries publications 521
- MQSeries Publish/Subscribe 458
- MQSeries server
 - connecting to 458
 - prerequisites 426
- multithreading
 - C 27
 - C++ 159
 - COBOL 240
 - Java 365

N

- Name attribute
 - distribution list 474
 - policy 475
 - publisher 474
 - service point 472
- name/value elements
 - C 24
 - C++ 157
 - COBOL 237
 - Java 362
- New Publications Only attribute 479

O

- OAMAS subset 28
- oamasami.h header 28
- object interface
 - overview 63
 - reference information 77
- object interface (COBOL)
 - overview 269
 - reference information 283
- object-style interface 13
- object-style interface (COBOL) 227
- objects
 - C 13
 - C++ 149
 - COBOL 227
 - Java 355
- open
 - AmDistributionList (C++) 205
 - AmDistributionList (Java) 403
 - AmPublisher (C++) 208
 - AmPublisher (Java) 406
 - AmReceiver (C++) 203
 - AmReceiver (Java) 401
 - AmSender (C++) 199
 - AmSender (Java) 397
 - AmSession (C++) 189
 - AmSession (Java) 388
 - AmSubscriber (C++) 210
 - AmSubscriber (Java) 408
- Open Shared attribute 478
- opening a session
 - C 16
 - C++ 151
 - COBOL 230
 - Java 357
- opening objects
 - C++ 151
 - Java 357
- operating systems 425
- OS/390
 - C applications 33

- OS/390 (*continued*)
 - COBOL applications 241
 - installation 440
 - prerequisites 425
- OS/390 subsystems, application
 - advice 421
- overloading
 - C++ 151
 - Java 357
- overview
 - C++ interface 167
 - C high-level interface 38
 - C object interface 63
 - COBOL high-level interface 244
 - COBOL object interface 269
 - Java interface 369

P

- PDF (Portable Document Format) 522
- Persistence attribute 476
- point-to-point 5
- poison messages 478
- policy
 - constants 509
 - defining 469
 - general attributes 476
 - initialization attributes 475
 - publish attributes 480
 - receive attributes 478
 - send attributes 476
 - subscribe attributes 479
 - summary 6
- policy interface
 - overview (C) 73
 - overview (C++) 178
 - overview (COBOL) 279
 - overview (Java) 379
- policy interface (C)
 - amPolClearErrorCodes 142
 - amPolGetLastError 142
 - amPolGetName 143
 - amPolGetWaitTime 143
 - amPolSetWaitTime 143
- policy interface (COBOL)
 - AMPOCLEC 348
 - AMPOGTLE 348
 - AMPOGTNA 349
 - AMPOGTWT 349
 - AMPOSTWT 350
- Portable Document Format (PDF) 522
- PostScript format 523
- prerequisites
 - compilers 426
 - disk space 425
 - MQSeries environment 426
 - OAMAS subset 28
 - operating systems 425
- Priority attribute 476
- problem determination 481
- problems, causes of 493
- procedural interface 13
- procedural interface (COBOL) 227
- programming languages 4
- publications
 - MQSeries 521

- publish
 - AmPublisher (C++) 208
 - AmPublisher (Java) 406
- Publish Locally attribute 480
- Publish On Request Only attribute 479
- publish/subscribe
 - constants 511
 - content-based 459
 - filters 459
 - introduction 5
 - using 458
- publish/subscribe messaging
 - C 22
 - C++ 156
 - COBOL 235
 - Java 361
- Publish To Others Only attribute 480
- publisher definition 474
- publisher interface
 - overview (C) 71
 - overview (C++) 176
 - overview (COBOL) 277
 - overview (Java) 377
- publisher interface (C)
 - amPubClearErrorCodes 131
 - amPubClose 131
 - amPubGetCCSID 131
 - amPubGetEncoding 132
 - amPubGetLastError 132
 - amPubGetName 133
 - amPubOpen 133
 - amPubPublish 134
- publisher interface (COBOL)
 - AMPBCL 337
 - AMPBCLC 337
 - AMPBGTEC 337
 - AMPBGTEN 338
 - AMPBGTE 338
 - AMPBGTNA 339
 - AMPBOP 339
 - AMPBPB 340

Q

- Queue Manager Name attribute 472
- Queue Name attribute 472

R

- readBytes
 - AmMessage (C++) 195
 - AmMessage (Java) 394
- reason codes
 - constants 512
 - description 497
- receive
 - AmReceiver (C++) 203, 204, 402
 - AmReceiver (Java) 401
 - AmSubscriber (C++) 210
 - AmSubscriber (Java) 408
- receiver definition 472
- receiver interface
 - overview (C) 69
 - overview (C++) 174
 - overview (COBOL) 275
 - overview (Java) 375
- receiver interface (C)
 - amRcvBrowse 115

- receiver interface (C) *(continued)*
 - amRcvBrowseSelect 117
 - amRcvClearErrorCodes 118
 - amRcvClose 119
 - amRcvGetDefnType 119
 - amRcvGetLastError 120
 - amRcvGetName 120
 - amRcvGetQueueName 121
 - amRcvOpen 121
 - amRcvReceive 122
 - amRcvReceiveFile 124
 - amRcvSetQueueName 125
- receiver interface (COBOL)
 - AMRCBR 322
 - AMRCBRSE 324
 - AMRCCL 326
 - AMRCCLEC 325
 - AMRCGTDI 326
 - AMRCGTLE 327
 - AMRCGTNA 327
 - AMRCGTQN 328
 - AMRCOP 328
 - AMRCRC 329
 - AMRCRCFL 330
 - AMRCSTQN 331
- Receiver Service attribute 474
- receiving files
 - C 21
 - C++ 155
 - COBOL 235
 - Java 361
- receiving messages
 - C 18
 - C++ 153
 - COBOL 232
 - Java 359
- reference information
 - C++ interface 183
 - C high-level interface 39
 - C object interface 77
 - COBOL high-level interface 246
 - COBOL object interface 283
 - Java interface 383
- Report Data attribute 476
- Report Type COA attribute 476
- Report Type COD attribute 476
- Report Type Exception attribute 476
- Report Type Expiry attribute 476
- repository, using
 - C 14
 - C++ 150
 - COBOL 228
 - Java 356
- repository cache (OS/390) 454
- repository file 450
- repository file (OS/390) 453
- request/response messaging
 - C 19
 - C++ 154
 - COBOL 234
 - Java 360
- reset
 - AmMessage (C++) 195
 - AmMessage (Java) 394
- Response Correl Id attribute 476
- Retain attribute 480
- Retry Count attribute 476
- Retry Interval attribute 476
- RF Header 472
- rollback
 - AmSession (C++) 189
 - AmSession (Java) 388
- runtime environment
 - AIX 429
 - AS/400 432
 - HP-UX 437
 - OS/390 440
 - Solaris 444
 - Windows 447

S

- sample programs
 - AS/400 461
 - OS/390 464
 - UNIX 461
 - Windows 461
- Segmentation attribute 476
- send
 - AmDistributionList (C++) 206
 - AmDistributionList (Java) 404
 - AmSender (C++) 199
 - AmSender (Java) 397
- sender definition 472
- sender interface
 - overview (C) 68
 - overview (C++) 173
 - overview (COBOL) 274
 - overview (Java) 374
- sender interface (C)
 - amSndClearErrorCodes 110
 - amSndClose 110
 - amSndGetCCSID 111
 - amSndGetEncoding 111
 - amSndGetLastError 111
 - amSndGetName 112
 - amSndOpen 112
 - amSndSend 113
 - amSndSendFile 114
- sender interface (COBOL)
 - AMSNCL 317
 - AMSNCLEC 316
 - AMSNGTCC 317
 - AMSNGTEN 317
 - AMSNGTLE 318
 - AMSNGTNA 318
 - AMSNOP 319
 - AMSNSEN 319
 - AMSNSENFL 320
- sendFile
 - AmDistributionList (C++) 206
 - AmDistributionList (Java) 404
 - AmSender (C++) 200
 - AmSender (Java) 398
- sending files
 - C 21
 - C++ 155
 - COBOL 235
 - Java 361
- sending group messages
 - C 26
 - C++ 159
 - COBOL 240
 - Java 365
- sending messages
 - C 16
 - C++ 152
 - COBOL 230
 - Java 357
- service
 - constants 509
 - defining 469
 - summary 4
- service point 472
- Service Type attribute 472
- session factory
 - overview (C++) 168
 - overview (Java) 370
- session interface
 - overview (C) 64
 - overview (C++) 169
 - overview (COBOL) 270
 - overview (Java) 371
- session interface (C)
 - amSesBegin 78
 - amSesClearErrorCodes 78
 - amSesClose 79
 - amSesCommit 79
 - amSesCreate 79
 - amSesCreateDistList 80
 - amSesCreateMessage 80
 - amSesCreatePolicy 80
 - amSesCreatePublisher 81
 - amSesCreateReceiver 81
 - amSesCreateSender 82
 - amSesCreateSubscriber 82
 - amSesDelete 83
 - amSesDeleteDistList 83
 - amSesDeleteMessage 83
 - amSesDeletePolicy 84
 - amSesDeletePublisher 84
 - amSesDeleteReceiver 84
 - amSesDeleteSender 85
 - amSesDeleteSubscriber 85
 - amSesGetDistListHandle 85
 - amSesGetLastError 86
 - amSesGetMessageHandle 86
 - amSesGetPolicyHandle 87
 - amSesGetPublisherHandle 87
 - amSesGetReceiverHandle 87
 - amSesGetSenderHandle 88
 - amSesGetSubscriberHandle 88
 - amSesOpen 88
 - amSesRollback 89
 - transaction coordination 78
 - unit of work 78
- session interface (COBOL)
 - AMSEBG 284
 - AMSECL 285
 - AMSECLEC 284
 - AMSECM 285
 - AMSECR 286
 - AMSECRDL 286
 - AMSECRMS 287
 - AMSECRPB 288
 - AMSECRPO 287
 - AMSECRRC 288
 - AMSECRSB 289
 - AMSECRSN 289
 - AMSEDL 290
 - AMSEDLDL 290

- session interface (COBOL) *(continued)*
 - AMSEDLMS 290
 - AMSEDLPB 291
 - AMSEDLPO 291
 - AMSEDLRC 291
 - AMSEDLNB 292
 - AMSEDLN 292
 - AMSEGHDL 292
 - AMSEGHMS 293
 - AMSEGHPB 294
 - AMSEGHPO 294
 - AMSEGHRC 294
 - AMSEGHSB 295
 - AMSEGHSN 295
 - AMSEGTLE 293
 - AMSEOP 296
 - AMSERB 296
 - transaction coordination 284
 - unit of work 284
- setCCSID
 - AmMessage (C++) 196
 - AmMessage (Java) 394
- setCorrelationId
 - AmMessage (C++) 196
 - AmMessage (Java) 394
- setDataOffset
 - AmMessage (C++) 196
 - AmMessage (Java) 394
- setElementCCSID
 - AmMessage (C++) 196
- setEncoding
 - AmMessage (C++) 196
 - AmMessage (Java) 395
- setFormat
 - AmMessage (C++) 197
 - AmMessage (Java) 395
- setGroupStatus
 - AmMessage (C++) 197
 - AmMessage (Java) 395
- setLocalHost
 - AmSessionFactory (C++) 185
 - AmSessionFactory (Java) 384
- setQueueName
 - AmReceiver (C++) 204
 - AmReceiver (Java) 402
 - AmSubscriber (C++) 210
 - AmSubscriber (Java) 408
- setRepository
 - AmSessionFactory (C++) 185
 - AmSessionFactory (Java) 385
- setTraceLevel
 - AmSessionFactory (C++) 185
 - AmSessionFactory (Java) 385
- setTraceLocation
 - AmSessionFactory (C++) 185
 - AmSessionFactory (Java) 385
- setVersion
 - AmElement (C++) 215
 - AmElement (Java) 412
- setWaitTime
 - AmPolicy (C++) 212
 - AmPolicy (Java) 410
- simulated group messages 26, 240
- Simulated Group Support attribute 472
- softcopy books 522
- Solaris
 - C++ applications 164

- Solaris *(continued)*
 - C applications 34
 - installation 443
 - Java applications 366
 - prerequisites 425
- Split File attribute 476
- string length constants 510
- structure of the AMI
 - C 13
 - C++ 149
 - COBOL 227
 - Java 355
- structure of this book xix
- structures, initial values 29
- subscribe
 - AmSubscriber (C++) 211
 - AmSubscriber (Java) 409
 - content-based 459
 - filters 459
- Subscribe Locally attribute 479
- subscriber definition 474
- subscriber interface
 - overview (C) 72
 - overview (C++) 177
 - overview (COBOL) 278
 - overview (Java) 378
- subscriber interface (C)
 - amSubClearErrorCodes 135
 - amSubClose 135
 - amSubGetCCSID 135
 - amSubGetDefnType 136
 - amSubGetEncoding 136
 - amSubGetLastError 137
 - amSubGetName 137
 - amSubGetQueueName 138
 - amSubOpen 138
 - amSubReceive 139
 - amSubSetQueueName 139
 - amSubSubscribe 140
 - amSubUnsubscribe 141
- subscriber interface (COBOL)
 - AMSBCL 341
 - AMSBCLEC 341
 - AMSBGTCC 342
 - AMSBGTD 342
 - AMSBGTEN 343
 - AMSBGTLE 343
 - AMSBGTNA 344
 - AMSBGTQN 344
 - AMSBOP 345
 - AMSBRC 345
 - AMSBRSB 346
 - AMSBSTQN 346
 - AMSBUN 347
- SupportPac 523
- Suppress Registration attribute 480
- Syncpoint attribute 476
- system default handle synonyms 515
- system default names 515
- system default objects
 - C 14
 - C++ 150
 - COBOL 228
 - Java 356

T

- terminology used in this book 519
- tool, administration 457
- topics, publish/subscribe
 - C 22
 - C++ 156
 - COBOL 235
 - Java 361
- toString
 - AmElement (C++) 215
 - AmElement (Java) 412
 - AmErrorException (C++) 221
 - AmErrorException (Java) 416
 - AmException (C++) 220
 - AmException (Java) 415
 - AmStatus (C++) 217
 - AmStatus (Java) 414
 - AmWarningException (C++) 222
 - AmWarningException (Java) 417
- trace
 - AS/400 482
 - C++ and Java 484
 - example 484
 - UNIX 482
 - using, OS/390 489
 - using, UNIX and Windows 481
 - Windows 483
- transaction coordination
 - C 78
 - C++ 186
 - COBOL 284
 - Java 386
- transaction support
 - C 26
 - C++ 158
 - COBOL 240
 - Java 364
- Trusted Option attribute 475

U

- Unicode character conversion 440
- unit of work
 - C 26, 78
 - C++ 158, 186
 - COBOL 240, 284
 - Java 364, 386
- UNIX
 - sample programs 461
- unsubscribe
 - AmSubscriber (C++) 211
 - AmSubscriber (Java) 409
- Unsubscribe All attribute 479
- Use Correl Id As Id attribute 479, 480
- using the AMI
 - C 13
 - C++ 149
 - COBOL 227
 - Java 355

W

- Wait For Whole Group attribute 478
- Wait Interval attribute 478
- Wait Interval Read Only attribute 478
- wait time constants 515

- what you need to know xix
- who this book is for xix
- Windows
 - C++ applications 165
 - C applications 35
 - installation 447
 - Java applications 366
 - prerequisites 425
 - sample programs 461
- Windows Help 523
- writeBytes
 - AmMessage (C++) 197
 - AmMessage (Java) 395
- writing applications
 - C 16
 - C++ 151
 - COBOL 230
 - Java 357
- writing applications for OS/390
 - subsystems 421
- writing IMS applications 421

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

User Technologies Department (MP095)
IBM United Kingdom Laboratories
Hursley Park
WINCHESTER,
Hampshire
SO21 2JN
United Kingdom

- By fax:
 - From outside the U.K., after your international access code use 44-1962-870229
 - From within the U.K., use 01962-870229
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
 - IBMLink™: HURSLEY(IDRCF)
 - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-5604-05

