Envoy Message Queuing

version 1.2.5

Envoy MQ for AS/400 Supplement to the Programmer's Guide

For use with Microsoft Message Queue services (MSMQ) software

ENVOY TECHNOLOGIES

Envoy Message Queuing

version 1.2.5

Envoy MQ for AS/400
Supplement to the
Programmer's Guide

Envoy Technologies Inc.
Corporate Headquarters
120 Wood Avenue South
Iselin, NJ, 08830, USA
Phone: 732-549-6500
Fax: 732-549-3165
Web: *http://www.envoytech.com*
Tech Support: *support@envoytech.com*

Envoy MQ for AS/400

Contents

1 - Installation 1

2 - ILE Programming 13

3 - RPG Interface 17

4 - COBOL Interface 53

Chapter 1

Installation

The Envoy MQ Client for AS/400 is the component of Envoy MQ running in the IBM AS/400 environment. The Envoy MQ Client communicates with Envoy MQ Connector, connecting your AS/400 applications to the MSMQ network.

The Envoy MQ Client for AS/400 is an extended version of Envoy MQ Client, which is described in the *Envoy Message Queuing Programmer's Guide*. The AS/400 Client is specially adapted for programming in IBM's Integrated Language Environment (ILE), in languages such as RPG and COBOL as well as C.

System and network requirements

You can install the Envoy MQ Client for AS/400 on an IBM AS/400 computer having the following minimum requirements:

- IBM OS/400 V3R2 or higher
- A TCP/IP communications link to at least one Windows NT system on which Envoy MQ Connector (version 1.2) is installed

If you intend to write and compile your own Envoy MQ Client applications, then you also need:

- An ILE compiler (C, C++, RPG, COBOL, or other ILE language)

To install the software from the Envoy MQ CD-ROM, you need:

- ❑ A Windows system with a CD-ROM drive and an FTP connection to the AS/400
- ❑ 5 Mb of free disk space for the Envoy MQ Client software

Installation procedure

The following instructions are to install Envoy MQ Client on AS/400 systems. You must also install Envoy MQ Connector on at least one Windows system in your network (for instructions, see the *Envoy MQ Connector Administrator's Guide*).

Where to install

You should install the Envoy MQ Client on each AS/400 system that you want to connect to MSMQ.

Installation file

The AS/400 client library is distributed in a compressed save (`SAVF`) file for AS/400 CISC (V3R2M0) and RISC (V3R7M0) systems. On the Envoy MQ installation CD-ROM, the save file is called `ENVOY MQ .LIB`. There are two versions of the file, which are located in:

`\clients\OS400\V3R2`	For OS/400 V3R2 and higher
`\clients\OS400\V3R7`	For OS/400 V3R7 and higher

The only significant difference between the versions is for RPG programming (see Chapter 3, *RPG Interface*). In the V3R7 version, you can use long RPG names for the Envoy MQ API identifiers. The long names are convenient because they are very similar to the identifiers in the native C-language API.

RPG for V3R2 does not support identifiers longer than 10 characters. The Envoy MQ version for V3R2 therefore uses short abbreviations for the RPG identifiers. See Appendix A, *RPG Interface for OS/400 V3R2*, for details.

Procedure

Please follow the instructions below to install the Envoy MQ Client on your AS/400.

1. From a Windows system with a CD-ROM drive and an FTP connection to the AS/400, start Telnet or another terminal emulation program.
2. Logon to the AS/400 using the `QSECOFR` user profile.

3. Run the following CL command to create an empty save (SAVF) file on the AS/400:

```
CRTSAVF FILE(QGPL/ENVOY MQ )
```

4. Insert the Envoy MQ CD-ROM in the drive.
5. Start an FTP program and send the save file to the AS/400. The following is an example for a command-line FTP client. The example assumes that the CD-ROM is in drive d:

```
OPEN <IP address of AS/400>
USER: QSECOFR
PASSWORD: ******
LCD d:\clients\OS400\V3Rx
CD QGPL
BINARY
PUT ENVOY MQ .LIB ENVOY MQ
```

6. Run the following command on the AS/400 to restore the library from the save file:

```
RSTLIB SAVLIB(ENVOY MQ ) DEV(*SAVF) SAVF(QGPL/ENVOY MQ )
+
MBROPT(*ALL) ALWOBJDIF(*ALL)
```

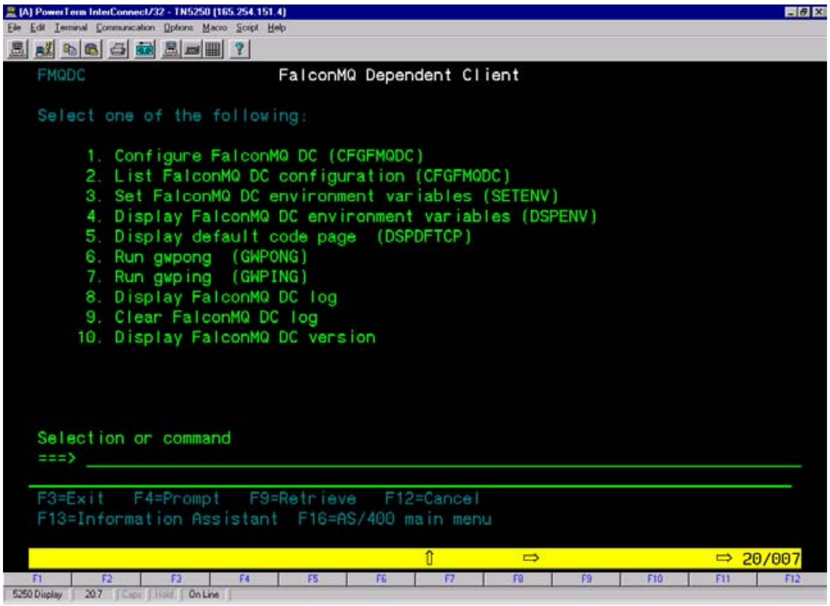
Configuration

Envoy MQ Client for AS/400 has a full-screen configuration editor. Before you run messaging applications, you should use the configuration editor to set parameters and options and to test the Envoy MQ Client/Server connection.

Displaying the Envoy MQ Client menu

To display the Envoy MQ Client menu, enter the following commands on the AS/400 system:

```
ADDLIBLE ENVOY MQ
GO FMQDC
```


[A] PowerTerm InterConnect/32 - TN5250 (165.254.151.4)
File Edit Terminal Communication Options Macro Script Help
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
5250 Display 20.7 Caps Hold On Line

```
FMQDC                   FalconMQ Dependent Client

Select one of the following:

     1. Configure FalconMQ DC (CFGFMQDC)
     2. List FalconMQ DC configuration (CFGFMQDC)
     3. Set FalconMQ DC environment variables (SETENV)
     4. Display FalconMQ DC environment variables (DSPENV)
     5. Display default code page  (DSPDFTCP)
     6. Run gwpong  (GWPONG)
     7. Run gwping  (GWPING)
     8. Display FalconMQ DC log
     9. Clear FalconMQ DC log
    10. Display FalconMQ DC version


Selection or command
===> ____________________________________________________

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
F13=Information Assistant  F16=AS/400 main menu

                                         20/007
```

✓ *In the menu, the abbreviation FalconMQ DC stands for Envoy MQ Client.*

From the menu, you may choose the following options. Alternatively, you can run an option by typing its name (`CFGFMQDC`, etc.) at the OS/400 command prompt.

Menu option	Description

`1. Configure FlaconMQ DC (CFGMQDC)`
`2. List FalconMQ DC configuration (CFGMQDC)`
	Defines connections with Envoy MQ Connector and code pages
	Displays the existing definitions

`3. Set FalconMQ DC environment variables (SETENV)`
`4. Display FalconMQ DC environment variables (DSPENV)`
	Sets or displays environment variables

`5. Display default code page (DSPDFTCP)`	Displays the name of the AS/400 system code page (the QCHRID system value)

Menu option	Description

`6. Run GWPONG (GWPONG)`
`7. Run GWPING (GWPING)`
	Runs an installation test of the Envoy MQ Client/Connector connection (see *Installation test* on page 11)

`8. Display FalconMQ DC log`
`9. Clear FalconMQ DC log`
	Displays the Envoy MQ error log
	Deletes the contents of the log

`10. Display FalconMQ DC version`	Displays Envoy MQ Client version information

Environment variables pointing to configuration files

To configure Envoy MQ Client, you need to create one or more configuration files on the AS/400 computer. The default configuration file is FMQ.ENV, located in the Envoy MQ Client library. Optionally, you can set the following environment variables to define the location of configuration files:

FMQROOT (Optional, default *LIBL) The library location of the default FMQ.ENV file. The value of FMQROOT should be the Envoy MQ Client library name.

FMQOVERRIDE (Optional) The location of an optional, secondary file that supplements and overrides the settings in the default FMQ.ENV file. Set FMQOVERRIDE to the library and filename, for example MYLIB/FMQ1.ENV.

The default FMQ.ENV file contains global default settings for all Envoy MQ applications on the computer. The FMQOVERRIDE file can contain supplementary settings for a particular user or application. For example, if FMQOVERRIDE contains additional Envoy MQ Connector connections, an application can connect to any of the Servers defined in either the default FMQ.ENV file or the FMQOVERRIDE file. In case of conflict between the settings in the files, the FMQOVERRIDE settings override the default FMQ.ENV.

The FMQOVERRIDE file is not required. If it is missing, the system takes all settings from the default FMQ.ENV file. Likewise, if a particular setting is missing from FMQOVERRIDE, the system takes the setting from the default

FMQ.ENV file. You can create any number of configuration files and switch between them by changing the value of FMQOVERRIDE.

Procedure

To set an environment variable, display the Envoy MQ Client menu and choose the option:

```
Set FalconMQ DC environment variables
```

Alternatively, you can run the SETENV command at the OS/400 prompt or embed the command in a CLP program.

For the FMQROOT variable, specify the name of the Envoy MQ Client library. For the FMQOVERRIDE variable, specify a library and filename. If the file doesn't exist, you can create it afterwards (see *Editing the configuration files* below).

To display the current settings, choose the option:

```
Display FalconMQ DC environment variables
```

✓ *For information about other Envoy MQ environment variables, see the* Installation *chapter in the* Envoy MQ Programmer's Guide.

Editing the configuration files

A configuration file contains the definitions of:

- ❑ Envoy MQ Connector connections, which Envoy MQ Client uses to transmit data to and from the Envoy MQ Connector
- ❑ Code-page translation tables, which Envoy MQ Client uses to convert EBCDIC string data to and from UNICODE

You can have any number of configuration files, but only two (the default FMQ.ENV file and the one identified by the FMQOVERRIDE environment variable) can be active at a time.

Each configuration file can contain any number of Envoy MQ Connector connection definitions and any number of code-page definitions.

Procedure

To edit a configuration file, display the Envoy MQ Client menu and choose the option:

```
Configure FalconMQ DC
```

At the top of the screen, specify the configuration file that you want to edit:

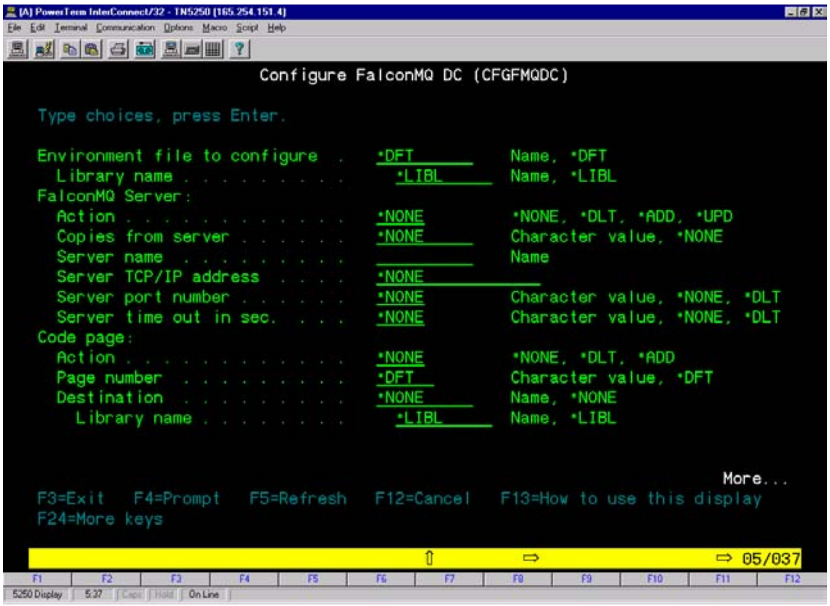
Environment to configure
Enter *DFT to configure the default FMQ.ENV file, or the name of a configuration file. If the file doesn't exist, it is created.

Library name
The location of the configuration file.

Follow the on-screen instructions to enter the other parameters and options (for more information, see *Explanation of configuration parameters* on page 8).

Press Enter to save your settings in the configuration file.

You can then enter a new set of options, and press Enter again. In this way, you can define multiple Server connection definitions and multiple code-page definitions in a single file.


[A] PowerTerm InterConnect/32 - TN5250 [165.254.151.4]
File Edit Terminal Communication Options Macro Script Help
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
5250 Display 5:37 Caps Hold On Line

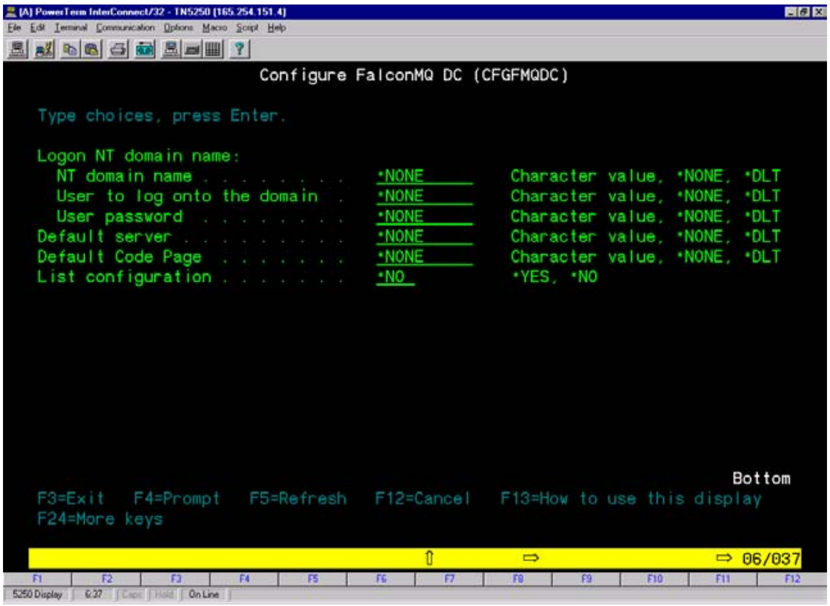
```
                       Configure FalconMQ DC (CFGFMQDC)

 Type choices, press Enter.

 Environment file to configure  .   *DFT__________   Name, *DFT
   Library name . . . . . . . . .     *LIBL________  Name, *LIBL
 FalconMQ Server:
   Action . . . . . . . . . . . .   *NONE            *NONE, *DLT, *ADD, *UPD
   Copies from server . . . . . .   *NONE_________   Character value, *NONE
   Server name  . . . . . . . . .   ______________   Name
   Server TCP/IP address  . . . .   *NONE_________________
   Server port number . . . . . .   *NONE            Character value, *NONE, *DLT
   Server time out in sec.  . . .   *NONE            Character value, *NONE, *DLT
 Code page:
   Action . . . . . . . . . . . .   *NONE            *NONE, *DLT, *ADD
   Page number  . . . . . . . . .   *DFT_____        Character value, *DFT
   Destination  . . . . . . . . .   *NONE_________   Name, *NONE
     Library name . . . . . . . .     *LIBL________  Name, *LIBL


                                                                      More...
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
                                         ⇧          ⇨               ⇨ 05/037
```

✓ *FalconMQ Server is the former designation of Envoy MQ Connector.*


(A) PowerTerm InterConnect/32 - TN5250 (165.254.151.4)
File Edit Terminal Communication Options Macro Script Help
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
5250 Display 6:37 Caps Hold On Line

```
                      Configure FalconMQ DC (CFGFMQDC)

 Type choices, press Enter.

 Logon NT domain name:
   NT domain name . . . . . . . .   *NONE         Character value, *NONE, *DLT
   User to log onto the domain  .   *NONE         Character value, *NONE, *DLT
   User password  . . . . . . . .   *NONE         Character value, *NONE, *DLT
 Default server . . . . . . . . .   *NONE         Character value, *NONE, *DLT
 Default Code Page  . . . . . . .   *NONE         Character value, *NONE, *DLT
 List configuration . . . . . . .   *NO           *YES, *NO










                                                                        Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys

                                    ⇧      ⇨                         ⇨ 06/037
```

✓ *The Envoy MQ Connector can run on any Windows server: NT, 2000 or XP..*

Displaying configuration settings

To review the complete set of configuration settings, display the Envoy MQ Client menu and choose the option:

```
List FalconMQ DC configuration
```

At the top of the screen, specify the configuration file that you want to display.

Command-line configuration utility

You can also configure Envoy MQ Client by running the `FMQDCCFG` command line configuration utility. For instructions, see the *Installation* chapter in the *Envoy MQ Programmer's Guide*.

Explanation of configuration parameters

The following paragraphs explain the parameters and options that you can set on configuration screen. For additional discussion and examples of the Envoy MQ Client configurations, see the *Installation* chapter of the *Envoy MQ Programmer's Guide*.

Envoy MQ Connector connection

The parameters in the FalconMQ Server section of the screen define a connection to a Envoy MQ Connector. Later, an application can connect to a Connector by specifying the connection name in the `FMQConnect` API function (see *Programming Messaging Applications* in the *Envoy MQ Programmer's Guide*). You can store any number of connections in a single configuration file.

`Action` Enter `*ADD` to add a new connection definition to the configuration file, `*UPD` to update a definition, or `*DLT` to delete a definition. Enter `*NONE` if you are not editing a connection definition.

`Copies from server` Where a parameter has not been explicitly defined for a connection, use the parameters of another connection as defaults (enter the second connection name).

`Server name` The connection name.

`Server TCP/IP address` IP address of the Envoy MQ Connector, or `*DLT` to delete an address that you previously entered.

`Server port number` TCP/IP port of the Server, or `*DLT` to delete a port that you previously entered. (default 1100).

`Server timeout in sec` TCP/IP timeout of the Client/Server connection in seconds, or `*DLT` to delete a timeout that you previously entered (default 30 seconds).

Code page

Envoy MQ automatically translates string-valued message properties (for example queue names) to UNICODE. For this to work, you need a UNICODE translation table for the code page that your application uses. Use the following switches to download code-page tables from Envoy MQ Connector and to manage the tables. (Before you do this, the code-page tables must be installed on the Server. See the *Installation* chapter of the *Envoy MQ Connector Administrator's Guide* for instructions.)

You can download any number of code-page tables. Envoy MQ Client uses the table for the code page that is in effect when your application runs (see the `Default code page` setting below).

In the `Code page` section of the configuration screen, enter the following options:

Action	Enter *ADD to download a code-page table, *DLT to delete a table that you previously downloaded, or *NONE if you are not editing a code-page definition.
Page number	The code-page number. Enter *DFT to download the default code page of the AS/400 system (the QCHRID system value).
Destination	An AS/400 filename to store the downloaded table.
Library name	The location to store the table.

Logon parameters

If you connect to Envoy MQ Connector using the explicit logon method, you must specify a Windows user name, password, and domain name. If you connect by the default login method, enter values of *NONE for all three parameters, or *DLT to delete a parameter that you previously entered.

✓ *For either logon method, you must also register a user in Windows (see the* Installation *chapter of the* Envoy MQ Connector Administrator's Guide *for an explanation of the logon methods and for instructions on user registration.*

In the Envoy MQ Connector section of the configuration screen, specify the Server name to which the logon parameters apply. You may use the same logon parameters for all Envoy MQ Connector connections, or different parameters for each connection.

Enter the following parameters in the Logon NT domain name section of the configuration screen.

NT domain name	Windows domain for Envoy MQ Connector logon.
User to log onto the domain	Windows user name.
User password	The Windows password.

The password that you enter here is actually only the suffix of the actual Windows password. The prefix is stored on the Envoy MQ Connector computer (see the *Envoy MQ Connector Administrator's Guide*). The password is stored in encrypted form.

Default settings

The following parameters specify default settings for Envoy MQ Client:

`Default server`	Enter the name of a Envoy MQ Connector connection that you previously defined. If a messaging application does not specify a connection, Envoy MQ Client connects to the default. Enter `*NONE` if you do not want to define a default, or `*DLT` to delete an existing default definition.
`Default code page`	Enter the number of a code page that you previously downloaded (see the `Code page` settings above). Envoy MQ Client uses the default code page to translate UNICODE to and from EBCDIC. Enter `*NONE` if you do not want to define a default code page, or `*DLT` to delete an existing default definition (in that case, Envoy MQ Client uses the system default, `QCHRID` value).

Installation test

To test the operation of Envoy MQ Client, run the `GWPING` and `GWPONG` programs supplied with the Envoy MQ software. These programs conduct a *ping-pong* test of the messaging system.

- The `GWPING` program sends *ping* messages via Envoy MQ Client and Envoy MQ Connector to a message queue.
- The `GWPONG` program sends *pong* replies to a second message queue, where it is read by `GWPING`.

✓ *Before you run the tests, you must define a default connection to Envoy MQ Connector and register the user name of the connection in Windows (for instructions, see* Configuration *on this book).*

Default test

To run a default test of communication from Envoy MQ Client to Envoy MQ Connector and back, follow these steps:

1. Enter the following commands to open the Envoy MQ Client menu:

```
ADDLIBLE ENVOY MQ
GO FMQDC
```

2. Choose the `Run GWPONG` option to start the `GWPONG` program Press Enter to accept the default test options.
3. Choose the `Run GWPING` option to start the `GWPING` program. Press Enter to accept the default test options.

The `GWPING` program sends a sequence of ten test messages, each containing the text `"PING"`, to a queue called `.\PongQ`. The `GWPONG` program waits to receive the messages, and then sends them back to a queue called `.\PingQ`. The `GWPING` program reads the replies from `.\PingQ` and signals you when they are received.

Results

For each of the ten test messages, `GWPING` should display *Ping sent* and *Received reply* together with the elapsed time.

In the event of an error, choose the `Display Envoy MQ DC log` option on the Envoy MQ Client menu to examine the error log. Review the installation and configuration of the Envoy MQ Client and Envoy MQ Connector.

Additional tests

You can set many test options on the `GWPING` and `GWPONG` screens. For an explanation of the options, see the *Installation* chapter of the *Envoy MQ Programmer's Guide*.

Chapter 2

ILE Programming

The Envoy MQ Client for AS/400 is supplied as an ILE service program. You can call the Envoy MQ Client API functions from programs written in any ILE language, for example C, C++, RPG, or COBOL.

This short chapter provides:

- ❑ Instructions for programming and binding Envoy MQ Client applications on the AS/400.
- ❑ Cross references to other Envoy MQ and MSMQ documentation, for details and examples of the API implementation.

API implementation

The native language of the Envoy MQ API is C. The API is identical to the C-language API of other Envoy MQ Clients, and nearly identical to the API of MSMQ. Thus you can port MSMQ or Envoy MQ Client applications very easily from other platforms to the AS/400.

The following references provide further information on the API:

- ❑ For programming information, please see the *Programming Messaging Applications* chapter in the *Envoy MQ Programmer's Guide*.
- ❑ For details of the API syntax, you should have a copy of the Microsoft MSMQ documentation and SDK online help.

Programming in C

Include the Envoy MQ `wintypes.h` and `mq.h` headers in your program. The header members are located in file `H` of the Envoy MQ Client library.

Compile your program using the IBM ILE C/400 compiler, for example:

```
CRTCMOD MODULE(YOURLIB/YOURMOD)
SRCFILE(YOURLIB/YOURPROG)
```

Source-code examples

For C source-code examples of Envoy MQ Client messaging applications, see the *Sample Application* chapter in the *Envoy MQ Programmer's Guide*. The source code of the `GWPING` and `GWPONG` programs is provided in the `SAMPLES` file of the Envoy MQ Client library.

Coding note for handles

If you set a handle to `NULL`, you should cast the `NULL` to the `HANDLE` type, for example:

```
HANDLE hConn = (HANDLE)NULL;
hRes = FMQDisconnect((HANDLE)NULL);
```

This comment applies to all Envoy MQ handles, for example security handles, connection handles, and queue handles.

Programming in languages other than C

You can call the C-language API from any ILE language. For example, you can program in a language such as IBM's ILE RPG/400 or ILE COBOL/400.

In practice, the API function calls involve some complex data structures. The structures are easy to create in C but may be difficult to translate into other languages. An easy solution to this problem is to program a small ILE module in C that handles the API calls.

Language interfaces supplied with Envoy MQ Client

If you program in RPG or COBOL, you can use one of the language interfaces that are supplied with Envoy MQ Client. These interfaces provide the definitions that you need to access the API. This solves the problem of translating the C syntax, so you can call the API functions directly in your RPG or COBOL code.

For details and source-code examples, see Chapter 3, *RPG Interface*, and Chapter 4, *COBOL Interface*.

Binding

Compile your program to an ILE module and bind it to the following Envoy MQ Client ILE service program:

```
ENVOY MQ /FMQDCLIB
```

For example, if your module is called YOURLIB/YOURMOD, issue the following command:

```
CRTPGM PGM(YOURLIB/YOURMOD) BNDSRVPGM(FAALCONMQ/FMQDCLIB) +
ACTGRP(*NEW)
```

The ACTGRP(*NEW) parameter is essential for Envoy MQ Client to function properly. It ensures initialization of the product static variables.

Chapter 3

RPG Interface

The Envoy MQ Client for AS/400 provides an RPG interface, which lets you call the Envoy MQ Client API functions directly from your RPG programs. The interface provides all the needed RPG definitions, so you can access the complete API without any C programming at all.

Versions for OS/400 V3R2+ and V3R7+

The RPG interface described in this chapter runs on OS/400 version V3R7 or higher. To use this interface, you must install the Envoy MQ Client version for V3R7 or higher (see the *Installation procedure* on page 2).

For a functionally identical interface that runs on OS/400 V3R2 and higher, see Appendix A, *RPG Interface for OS/400 V3R2*.

The only significant difference between the two interfaces is the length of the API identifiers. RPG for V3R2 supports identifiers of up to 10 characters. RPG for V3R7 supports long identifiers, which are more similar to the C-language identifiers in the native Envoy MQ and MSMQ APIs and are more convenient to use.

Overview of the interface

The interface is implemented in an RPG copy member called `FMQCONST`. This chapter explains:

- ❑ The steps for creating a Envoy MQ Client application in RPG
- ❑ The structure and contents of `FMQCONST`
- ❑ Techniques for calling the Envoy MQ Client API functions using the definitions in `FMQCONST`

The interface provides two additional copy members, called `FMQPROPVAR` and `FMQLOCATE`, which support dynamic programming techniques for building message and queue property structures. The chapter includes:

- ❑ Sample RPG data structures representing MSMQ message and queue properties, constructed using static or dynamic techniques
- ❑ Sample RPG messaging applications

API functions This chapter describes an interface that you can use to call the Envoy MQ Client API functions in RPG programs. It does not document the API functions themselves. For information on that subject, see *API implementation* on page 13 and the references therein.

Programming steps

To program a Envoy MQ Client messaging application, follow these steps:

1. Copy the `FMQCONST` member, which is found in the `QRPGLESRC` file of the Envoy MQ Client library, into the definition specifications of your RPG program (see *FMQCONST copy member* on page 19).
2. Optionally, copy the `FMQPROPVAR` and/or `FMQLOCATE` members into the definition specifications. These members can help you set up the data structures you need for Envoy MQ Client API calls (see *Copy members* on page 49).
3. Create RPG definitions for the required message and queue properties (see *Data structures (static method)* on page 30 or *Data structures (dynamic method)* on page 38).
4. Code the Envoy MQ Client API calls (see *Sample program* on page 44).
5. Compile the program to an ILE module using the IBM ILE RPG/400 compiler.
6. Bind the ILE module to the following Envoy MQ ILE service program:

```
ENVOY MQ /FMQDCLIB
```

For example, if your module is called `YOURLIB/YOURMOD`, issue the following command:

```
CRTPGM PGM(YOURLIB/YOURMOD) +
MODULE(YOURLIB/YOURMOD) BNDSRVPGM(FALCONMQ/FMQDCLIB)
ACTGRP(*NEW)
```

FMQCONST copy member

The FMQCONST copy member provides the definitions that you need to access the Envoy MQ Client API. You must copy FMQCONST into the definition specifications of your RPG program. FMQCONST is found in the QRPGLESRC file of the Envoy MQ Client library.

The FMQCONST definitions include:

- ❑ Constants representing message properties
- ❑ Constants representing queue properties
- ❑ Constants representing queue manager properties
- ❑ Constants representing the value types of properties
- ❑ Miscellaneous named constants
- ❑ Declarations and arguments of the API functions

In general, the definitions are identical to the C-language definitions in the C header files, `mq.h`, `wintypes.h`, and `fmqpubd.h`, which are also supplied with Envoy MQ Client. The main exception to this rule is that the names of the constants are abbreviated relative to the C versions, according to the requirements of the RPG syntax. For example, the C constant `PROPID_M_DEST_QUEUE_LEN` (representing the message property *destination queue name length*) is abbreviated to `PID_M_DEST_LEN` in RPG.

✓ *The following tables list the most important identifiers in the RPG interface. Please refer to the `FMQCONST` source code for other identifiers not listed in the tables.*

Message properties

The following table lists the constants representing message properties in RPG and their equivalents in C.

The table also lists the following information, which is needed to construct a propvariant structure for each property (see *Substructures of property structures* on page 33):

- ❑ The value type constant of the property in RPG (for a list of the corresponding constants in C, see *Value type constants* on page 24)
- ❑ The data type of the property value

✓ *The* VT_NULL *value types are permitted only when receiving a message. See the Microsoft MSMQ documentation for complete details about the meaning of each property and the permitted values and types.*

RPG			**Equivalent in C**
Message property	**Value type**	**Data type**	**Message property**
PID_M_ACK	VT_UI1 (or VT_NULL)	1A	PROPID_M_ACKNOWLEDGE
PID_M_ADMIN_Q	VT_LPWSTR	*	PROPID_M_ADMIN_QUEUE
PID_M_ADMQ_LEN	VT_UI4	10U 0	PROPID_M_ADMIN_QUEUE_LEN
PID_M_APPSPC	VT_UI4 (or VT_NULL)	10U 0	PROPID_M_APPSPECIFIC
PID_M_ARVTIME	VT_UI4 (or VT_NULL)	10U 0	PROPID_M_ARRIVEDTIME
PID_M_AUTH_LVL	VT_UI4	10U 0	PROPID_M_AUTH_LEVEL
PID_M_AUTHTCAT	VT_UI1 (or VT_NULL)	1A	PROPID_M_AUTHENTICATED
PID_M_BODY	VT_VECTOR#UI1	Two fields: 10U 0 *	PROPID_M_BODY
PID_M_BODY_LEN	VT_UI4 (or VT_NULL)	10U 0	PROPID_M_BODY_SIZE
PID_M_BODY_TYP	VT_UI4 (or VT_NULL)	10U 0	PROPID_M_BODY_TYPE
PID_M_CERT_LEN	VT_UI4	10U 0	PROPID_M_SENDER_CERT_LEN
PID_M_CLASS	VT_UI2 (or VT_NULL)	5I 0	PROPID_M_CLASS
PID_M_CONN_TYP	VT_CLSID	*	PROPID_M_CONNECTOR_TYPE
PID_M_CORRID	VT_VECTOR#UI1	Two fields: 10U 0 *	PROPID_M_CORRELATIONID
PID_M_DELIVERY	VT_UI1 (or VT_NULL)	10U	PROPID_M_DELIVERY

RPG			Equivalent in C
Message property	Value type	Data type	Message property
PID_M_DEST_LEN	VT_UI4 (or VT_NULL)	10U 0	PROPID_M_DEST_QUEUE_LEN
PID_M_DEST_Q	VT_LPWSTR	*	PROPID_M_DEST_QUEUE
PID_M_ENCR_ALG	VT_UI4	10U 0	PROPID_M_ENCRYPTION_ALG
PID_M_EXT	VT_VECTOR#UI1	Two fields: 10U 0 *	PROPID_M_EXTENSION
PID_M_EXT_LEN	VT_UI4	10U 0	PROPID_M_EXTENSION_LEN
PID_M_HASH_ALG	VT_UI4	10U 0	PROPID_M_HASH_ALG
PID_M_JOURNAL	VT_UI1	1A	PROPID_M_JOURNAL
PID_M_LABEL	VT_LPWSTR	*	PROPID_M_LABEL
PID_M_LBL_LEN	VT_UI4	10U 0	PROPID_M_LABEL_LEN
PID_M_MSGID	VT_VECTOR#UI1	Two fields: 10U 0 *	PROPID_M_MSGID
PID_M_PRIORITY	VT_UI1 (or VT_NULL)	1A	PROPID_M_PRIORITY
PID_M_PRIV_LVL	VT_UI4 (or VT_NULL)	10U 0	PROPID_M_PRIV_LEVEL
PID_M_PROV_TYP	VT_UI4	10U 0	PROPID_M_PROV_TYPE
PID_M_PROVN	VT_LPWSTR	*	PROPID_M_PROV_NAME
PID_M_PROVN_LN	VT_UI4 (or VT_NULL)	10U 0	PROPID_M_PROV_NAME_LEN
PID_M_RES_Q	VT_LPWSTR	*	PROPID_M_RESP_QUEUE
PID_M_RESQ_LEN	VT_UI4	10U 0	PROPID_M_RESP_QUEUE_LEN
PID_M_SEC_CNTX	VT_UI4	10U 0	PROPID_M_SECURITY_CONTEXT
PID_M_SENDERID	VT_VECTOR#UI1	Two fields: 10U 0 *	PROPID_M_SENDERID

RPG			Equivalent in C
Message property	Value type	Data type	Message property
PID_M_SENTTIME	VT_UI4 (or VT_NULL)	10U 0	PROPID_M_SENTTIME
PID_M_SID_LEN	VT_UI4	10U 0	PROPID_M_SENDERID_LEN
PID_M_SID_TYPE	VT_UI4 (or VT_NULL)	10U 0	PROPID_M_SENDERID_TYPE
PID_M_SIGN	VT_VECTOR#UI1	Two fields: 10U 0 *	PROPID_M_SIGNATURE
PID_M_SIGN_LEN	VT_UI4	10U 0	PROPID_M_SIGNATURE_LEN
PID_M_SKEY	VT_VECTOR#UI1	Two fields: 10U 0 *	PROPID_M_DEST_SYMM_KEY
PID_M_SKEY_LEN	VT_UI4	10U 0	PROPID_M_DEST_SYMM_KEY_LEN
PID_M_SMCH_ID	VT_CLSID	*	PROPID_M_SRC_MACHINE_ID
PID_M_SNDR_CRT	VT_VECTOR#UI1	Two fields: 10U 0 *	PROPID_M_SENDER_CERT
PID_M_T2ARV	VT_UI4	10U 0	PROPID_M_TIME_TO_REACH_QUEUE
PID_M_T2RCV	VT_UI4 (or VT_NULL)	10U 0	PROPID_M_TIME_TO_BE_RECEIVED
PID_M_TRACE	VT_UI1 (or VT_NULL)	1A	PROPID_M_TRACE
PID_M_VERSION	VT_UI4	10U 0	PROPID_M_VERSION
PID_M_XSTS_Q	VT_LPWSTR	*	PROPID_M_XACT_STATUS_QUEUE
PID_M_XSTS_QLN	VT_UI4	10U 0	PROPID_M_XACT_STATUS_QUEUE_LEN

Queue properties

The following table lists the constants representing queue properties in RPG and their equivalents in C.

RPG			Equivalent in C
Queue property	Value type	Data type	Queue property
PID_Q_AUTHNCTE	VT_UI1	1A	PROPID_Q_AUTHENTICATE
PID_Q_BASEPRIO	VT_I2	5I 0	PROPID_Q_BASEPRIORITY
PID_Q_CHGTIME	VT_I4	10I 0	PROPID_Q_MODIFY_TIME
PID_Q_CRTIME	VT_I4	10I 0	PROPID_Q_CREATE_TIME
PID_Q_INSTNC	VT_CLSID	*	PROPID_Q_INSTANCE
PID_Q_JRN	VT_UI1	1A	PROPID_Q_JOURNAL
PID_Q_JRQUOTA	VT_UI4	10U 0	PROPID_Q_JOURNAL_QUOTA
PID_Q_LABEL	VT_LPWSTR	*	PROPID_Q_LABEL
PID_Q_PATH	VT_LPWSTR	*	PROPID_Q_PATHNAME
PID_Q_PRIVLVL	VT_UI4	10U 0	PROPID_Q_PRIV_LEVEL
PID_Q_QUOTA	VT_UI4	10U 0	PROPID_Q_QUOTA
PID_Q_TYPE	VT_CLSID	*	PROPID_Q_TYPE
PID_Q_XACT	VT_UI1	1A	PROPID_Q_TRANSACTION

Queue manager properties

The following table lists the constants representing queue manager properties in RPG and their equivalents in C.

RPG			Equivalent in C
Queue manager property	Value type	Data type	Queue manager property
PID_QM_CONNECT	VT_VECT#LPWSTR	Two fields: 10U 0 *	PROPID_QM_CONNECTION
PID_QM_ENCRYPT	VT_VECTOR#UI1	Two fields: 10U 0 *	PROPID_QM_ENCRYPTION_PK
PID_QM_MCH_ID	VT_CLSID	*	PROPID_QM_MACHINE_ID
PID_QM_PATH	VT_LPWSTR	*	PROPID_QM_PATHNAME
PID_QM_SITE_ID	VT_CLSID	*	PROPID_QM_SITE_ID

Value type constants

The following table lists the value type constants defined in FMQCONST and the corresponding constants defined in the C header files. Only the constants that are currently used in MSMQ are listed.

The value types are used in propvariant structures, which store the values of properties. For a full explanation, see *Substructures of property structures* on page 33. For reference, the table also indicates:

- ❑ The data types of the value fields in a propvariant structure
- ❑ The interpretation of the value fields
- ❑ The names of the corresponding value fields in C

RPG			Equivalent in C		
Value type constant	Data type	Interpretation of property value	Value type constant	Data type	Union field name
VT_CLSID	*	Base pointer (points to a GUID code, type 16A)	VT_CLSID	CLSID _RPC_FAR	*puuid
VT_I2	5I 0	Property value	VT_I2	short	iVal

RPG			Equivalent in C		
Value type constant	Data type	Interpretation of property value	Value type constant	Data type	Union field name
VT_I4	10I 0	Property value	VT_I4	long	lVal
VT_LPWSTR	*	Base pointer (points to a null-terminated string)	VT_LPWSTR	LPWSTR	pwszVal
VT_NULL		No value (permitted only when receiving a message)	VT_NULL		
VT_UI1	1A	Property value	VT_UI1	UCHAR	bVal
VT_UI2	5I 0	Property value	VT_UI2	USHORT	uiVal
VT_UI4	10U 0	Property value	VT_UI4	ULONG	ulVal
VT_VECT#LPWSTR	Two fields: 10U 0 *	 Length of buffer Base pointer (points to buffer)[a]	VT_VECTOR | VT_LPWSTR	CALPWSTR	calpwstr
VT_VECTOR#UI1	Two fields: 10U 0 *	 Length of buffer Base pointer (points to buffer)[b]	VT_VECTOR | VT_UI1	CAUI1	caub

Notes

a. For the value type VT_VECT#LPWSTR, the buffer contains a null-terminated string.

b. For the value type VT_VECTOR#UI1, the buffer may contain various types of binary or text data:

- The message body property (PID_M_BODY) has this value type and may contain any data whatsoever.
- Other properties having this value type are restricted to certain types or structures of data. For information about specific properties, see the Microsoft MSMQ documentation and SDK online help.

Miscellaneous named constants

FMQCONST defines a large number of constants representing special values of API function arguments, error codes, etc. The following are a few examples:

Constant in RPG	Equivalent in C
MQ_ACCESS_ALL	PSD_SPECIALACCESS_ALL
MQ_ER_ACCESS	MQ_ERROR_ACCESS_DENIED
MQ_ER_BUF_OVR	MQ_ERROR_BUFFER_OVERFLOW
MQ_LE	PRLE

✓ *The constants are too numerous to list here. For a complete listing, please refer to the FMQCONST source code.*

API functions

FMQCONST provides a complete set of definitions for the Envoy MQ Client API functions. The functions are defined as external procedures in RPG.

Function example

The following is the definition of the external procedure MQSendMessage in FMQCONST. The procedure is equivalent to the MQSendMessage() function in the MSMQ or Envoy MQ Client API.

```
     D MQSendMessage    PR            10U 0 EXTPROC('MQSendMessage')
     D  QueueHandle                   10U 0 VALUE
     D  MsgProps                        *   VALUE
     D  ITransact                     16A   VALUE
```

The function accepts three parameters by value:

- A queue handle of type U(10,0), specifying the destination queue.
- A base pointer of type *(16), pointing to a message property structure containing the content of the message.

- ❑ A transaction handle of type A(16), specifying a transaction to which the message belongs (optionally `NULL`).

The function returns a result code of type U(10,0).

Calling syntax

In the calculation specifications of your program, you can call the `MQSendMessage` procedure using syntax such as the following:

```
C                  EVAL      hRes = MQSendMessage(Q1_Handle          :
C                                                 %ADDR(M1_MsgProps) :
C                                                 MQ_NO_XACT)
```

For the method of setting up the message property structure (`M1_MsgProps`), see *Data structures (static method)* on page 30 or *Data structures (dynamic method)* on page 38. For other examples of function calls, see the *Sample program* on page 44.

Comparison with C

For comparison, the following is the corresponding API function declaration in C:

```
HRESULT APIENTRY MQSendMessage(
  QUEUEHANDLE hDestinationQueue,
  MQMSGPROPS * pMessageProps,
  ITransaction * pTransaction
);
```

Notice that the names of the parameters in RPG are abbreviated from the argument names or data types in C.

List of Envoy MQ Client functions

The following is a list of Envoy MQ Client API functions. The names of the RPG procedures are the same as the C function names. For the complete RPG definition of each procedure, please see the `FMQCONST` source code.

For an explanation of the procedures and their parameters, please see the following references:

A. The chapter on *Programming Messaging Applications* in the *Envoy MQ Programmer's Guide.*

B. The Microsoft MSMQ documentation and SDK online help.

Procedure	**Reference**
FMQAbort	A
FMQCommit	A
FMQConnect	A
FMQDebug	A
FMQDisconnect	A
FMQGetLogPath	A
FMQSetLogPath	A
FMQVersion	A
FMQV1Connect()	A
MQBeginTransaction	A
MQCloseCursor	B
MQCloseQueue	B
MQCreateCursor	B
MQCreateQueue	A, B
MQDeleteQueue	B

Procedure	Reference
MQFreeMemory	B
MQFreeSecurityContext	A, B
MQGetMachineProperties	B
MQGetQueueProperties	B
MQGetSecurityContext	A, B
MQHandleToFormatName	B
MQInstanceToFormatName	B
MQLocateBegin	A, B
MQLocateEnd	B
MQLocateNext	B
MQOpenQueue	B
MQPathNameToFormatName	B
MQReceiveMessage	A, B
MQRegisterCertificate	A, B
MQSendMessage	B
MQSetQueueProperties	B

Short API function names

For compatibility with Envoy MQ Client version 1.0, the following shorter names for API functions are also supported. Please refrain from using the short names because they may be discontinued in future versions.

Procedure	Short name (version 1.0)
MQBeginTransaction	MQBeginTrnsact
MQFreeSecurityContext	MQFreSecContxt
MQGetMachineProperties	MQGetMchProp
MQGetQueueProperties	MQGetQueueProp
MQGetSecurityContext	MQGetSecContxt
MQHandleToFormatName	MQHndlToFormat

Procedure	Short name (version 1.0)
`MQInstanceToFormatName`	`MQInstToFormat`
`MQPathNameToFormatName`	`MQPathToFormat`
`MQReceiveMessage`	`MQRcvMessage`
`MQSetQueueProperties`	`MQSetQueueProp`

Data structures (static method)

Many of the MSMQ and Envoy MQ Client API functions require parameters that are pointers to data structures. These include:

Property structures	Structures containing sets of message, queue, or queue manager properties. The content of a message, for example, is specified in a message property structure.
Substructures of property structures	Structures and arrays that are elements of property structures. An example is the *propvariant structure*, which contains the values of properties.
Query structures	Structures required as parameters of the `MQLocateBegin` function, which searches for queues having specified property values.

This section explains a simple, *static* method to create the property structures and substructures in your RPG programs. Most of the examples are taken from the *Sample program*, which is presented in full on page 44. If you wish, you can copy the examples (with minor modifications) into your RPG programs.

For additional information on the interpretation and use of the structures, please refer to the Microsoft MSMQ documentation and SDK online help.

For information on the query structures, please see the *Online samples* described on page 48.

Static method

In the static method, a property structure specifies a fixed set of properties.

For example, you can create a static message property structure containing the message body, delivery, and priority properties. Every message that you send using this property structure contains exactly these three properties.

You need to define separate property structures for message properties and for queue properties. Optionally, you can define multiple static structures for different sets of message or queue properties.

In an RPG program, you can implement a static property structure using simple data structure (`DS`) definitions. Often, you can leave the `DS` subfields unnamed and initialize their values in the definition. Depending on the needs of your application, you can also name the fields and assign or read the values in the calculation specifications.

The examples in the following sections illustrate the static method. For information on the alternative dynamic method, which lets you store a varying set of properties in a single structure, see *Data structures (dynamic method)* on page 38.

Property structures

A *property structure* contains a collection of properties and their values. There are three types of property structures, each of which is equivalent to one of the data structures defined in the C header files of the MSMQ API.

Structure	Contains a collection of	Equivalent to C data type
Message property structure	Message properties	`MQMSGPROPS`
Queue property structure	Queue properties	`MQQUEUEPROPS`
Queue manager property structure	Queue manager properties	`MQQMPROPS`

Each property structure contains the following four fields:

RPG data type	C data type	Field name in C	Description
`10U 0`	`DWORD`	`cProp`	A count of the properties included in the structure. The value of this field is the size of the arrays in the other fields of the structure.
`*`	Array of `PROPID`	`aPropID`	A pointer to an array of `PID_...` constants, identifying the properties that are included in the structure (input to the API functions).
`*`	Array of `PROPVARIANT`	`aPropVar`	A pointer to an array of propvariant structures, which contain the values of the properties (input or output).
`*`	Array of `HRESULT`	`aStatus`	A pointer to an array of status codes (output from the API functions).

✓ *For convenience, we sometimes refer to the fields by the generic names `cProp`, `aPropID`, etc. In RPG, you must use field names that are unique throughout the entire program. You can leave the fields unnamed if the program doesn't need to change their values.*

Message properties

The following is a sample definition of a message property structure. The structure is initialized to contain three message properties. The fields are named with the prefix `M1_...`, on the assumption that you may define more than one such structure (`M2_...`, `M3_...`, etc.) in your program. For example, you could use `M1_MsgProps` for messages that you send and `M2_MsgProps` for messages that you receive.

For the definitions of the array fields (`M1_Props`, `M1_Values`, and `M1_Status`), see *Substructures of property structures* on page 33.

```
      * Message property structure
     D M1_MsgProps     DS
     D  M1_cProp                     10U 0 INZ(3)
     D  M1_aPropID                     *   INZ(%ADDR(M1_Props))
     D  M1_aPropVar                    *   INZ(%ADDR(M1_Values))
     D  M1_aStatus                     *   INZ(%ADDR(M1_Status))
```

Queue properties

A queue property structure is defined in exactly the same way as a message property structure. In the following example, the fields of the structure are unnamed, so they cannot be changed from their initial values.

```
 * Queue property structure
D Q_QProps        DS
D                               10U 0 INZ(3)
D                                 *   INZ(%ADDR(Q_Props))
D                                 *   INZ(%ADDR(Q_Values))
D                                 *   INZ(%ADDR(Q_Status))
```

Queue manager properties

Queue manager property structures are completely analogous to message and queue property structures. The following is a sample definition containing a single queue manager property.

```
 * Queue manager property structure
D QMProps         DS
D                               10U 0 INZ(1)
D                                 *   INZ(%ADDR(QM_Props))
D                                 *   INZ(%ADDR(QM_Values))
D                                 *   INZ(%ADDR(QM_Status))
```

Substructures of property structures

Each property structure (see *Property structures* on page 31) contains pointers to three arrays:

`aPropID` Pointer to an array of property identifies (`PID_...` constants) identifying message, queue, or queue manger properties.

`aPropVar` Pointer to an array of propvariant structures, which contain the values of the properties.

`aStatus` Pointer to an array of status codes, used for output from the API functions.

The number of elements in each array is given by the `cProp` field of the property structure. The order of properties must be identical in each array. For example, if the `aPropID` array contains `PID_...` constants for the message body, delivery, and priority properties, then the other arrays must also contain elements for exactly the same properties, in the same order.

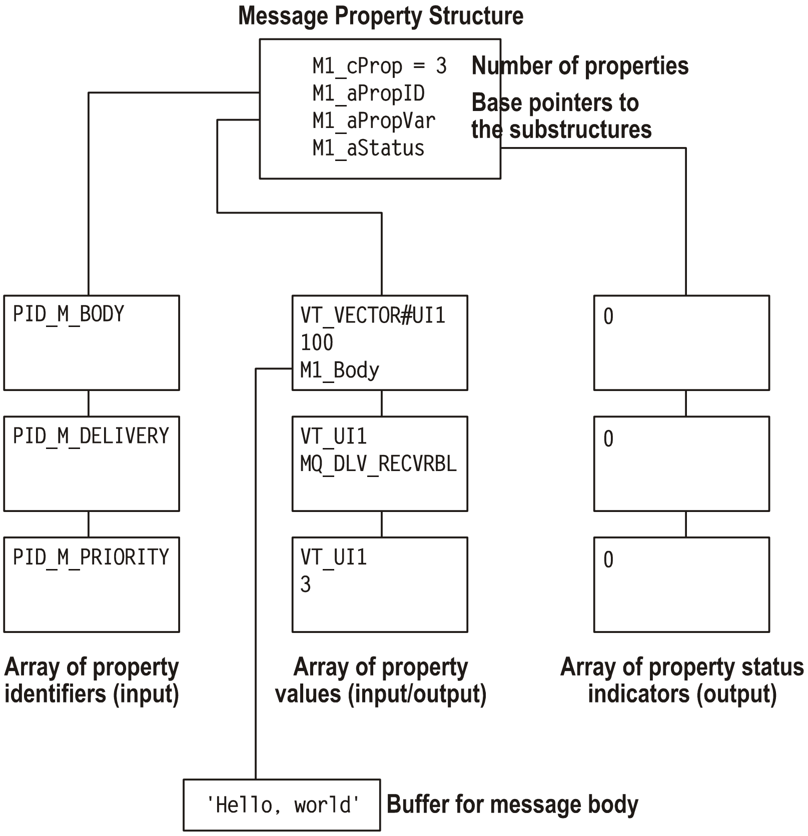
The following examples illustrate how you can construct the arrays in an RPG program. For convenience, the arrays are represented as RPG data structures (in essence, substructures of a property structure) instead of true RPG arrays.

The examples are for the message and queue property structures, M1_MsgProps and Q_QProps, which are defined in the *Property structures* section on page 31.

There are three properties in each message array, as specified by the cProp field of M1_MsgProps. The properties included in the example are:

- ❑ Message body
- ❑ Message delivery
- ❑ Message priority

There are three properties in each queue array, as specified by the cProp field of Q_QProps:



- ❑ Queue path name
- ❑ Queue label
- ❑ Queue transaction status (transacted or nontransacted queue)

In a single property structure, the number and order of properties must be identical in each array.

Array of property identifiers

The following data structure represents an array of `PID` constants identifying the properties included in a message property structure. The structure corresponds to the `aProp` field of a C property structure.

```
 * aProp array of message property identifiers
D M1_Props        DS
D                               10U 0 INZ(PID_M_BODY)
D                               10U 0 INZ(PID_M_DELIVERY)
D                               10U 0 INZ(PID_M_PRIORITY)
```

Arrays of queue and queue manager properties are defined in exactly the same way, for example:

```
 * aProp array of queue property identifiers
D Q_Props         DS
D                               10U 0 INZ(PID_Q_PATH)
D                               10U 0 INZ(PID_Q_LABEL)
D                               10U 0 INZ(PID_Q_XACT)
```

Array of propvariant structures

MSMQ and Envoy MQ Client use propvariant structures to store the values of message, queue, and queue manager properties. On the AS/400, a propvariant is a 48-byte structure containing the following fields:

Value type constant A `VT_...` constant indicating the data type of the property value.

Reserved Reserved for future use.

Value1 The value of the property. For certain properties, *Value1* is the size of the value in bytes (equivalent to the `cElems` field in C).

Value2 If *Value1* contains the value, *Value2* is an empty placeholder field. If *Value1* contains the size of the value, then *Value2* is a pointer to the value (equivalent to the `pElems` field in C).

The following example is a data structure containing three propvariant substructures. The propvariant elements contain the values of the message body, delivery, and priority properties, respectively. The structure as a whole corresponds to the aPropVar array of a C message property structure.

```
 * aPropVar array of message property values
D M1_Values       DS
 *
 * Propvariant structure specifying the message body
 * (The body is stored in a 100-byte buffer M1_Body)
D                                5U 0 INZ(VT_VECTOR#UI1)
D                               14A   INZ(MQ_Reserved)
D                               10U 0 INZ(100)
D                                 *   INZ(%ADDR(M1_Body))
 *
 * Propvariant structure specifying the message delivery
 * (MQ_DLV_RECVRBL means recoverable delivery, guaranteed even after
 * recovery from a crash)
D                                5U 0 INZ(VT_UI1)
D                               14A   INZ(MQ_Reserved)
D                                2B 0 INZ(MQ_DLV_RECVRBL)
D                                 *
 *
 * Propvariant structure specifying a message priority of 3
D                                5U 0 INZ(VT_UI1)
D                               14A   INZ(MQ_Reserved)
D                                2B 0 INZ(3)
D                                     *
```

In each propvariant, the value type constant and the data types of *Value1* and *Value2* are set according to the property whose value is stored. To determine the correct value and data types, refer to the tables in the *FMQCONST copy member* section, starting on page 19.

For example, the first propvariant stores the value of the message body property, PID_M_BODY. Referring to the *Message properties* table on page 19, the value type constant for PID_M_BODY is VT_VECTOR#UI1. The data type of *Value1* is 10U 0, and the data type of *Value2* is *. *Value1* stores the length of the message buffer, and *Value2* is a pointer to the message buffer M1_Body.

Elsewhere in the program, you need to define the message buffer and store a message in it, for example:

```
 * Content of the message body
D M1_Body         S            100A   INZ('Hello, world')
```

✓ *In the above example,* Value1 *for the message body is set to the full length of the message buffer (100 bytes). Depending on how your application interprets the contents of the message buffer, you may need to set* Value1 *to the true length of the message stored in the buffer (*`'Hello, world'` *= 12 bytes, or 13 bytes if the string is null terminated).*

The `aPropVar` array of queue property values is constructed according to the same principles. The following code is an example.

```
 * aPropVar array of queue property values
D Q_Values        DS
 *
 * Propvariant structure specifying the queue path name
D                                5U 0 INZ(VT_LPWSTR)
D                               14A   INZ(MQ_Reserved)
D  pQPath                         *   INZ(%ADDR(Q_Path))
D                                 *
 *
 * Propvariant structure specifying the queue label
D                                5U 0 INZ(VT_LPWSTR)
D                               14A   INZ(MQ_Reserved)
D  pQLabel                        *   INZ(%ADDR(Q_Label))
D                                 *
 *
 * Propvariant structure specifying the queue transaction status
D                                5U 0 INZ(VT_UI1)
D                               14A   INZ(MQ_Reserved)
D  Q_Xact                        1A
D                                 *
```

Elsewhere in the program, you need to define buffers for the queue path name and label:

```
 * Buffers for the queue path name and label
D Q_Path          S             50A
D Q_Label         S            100A
```

Array of status codes

The following example is structure of message status codes. The structure corresponds to an `aStatus` array in a C message property structure.

The status codes are output from various API functions. In the example, the three codes are given names (`M1_Body_sts`, etc.) so the program can retrieve the output values.

```
      * aStatus array of message-property status codes
     D M1_Status       DS                  INZ
     D  M1_Body_sts                  10U 0
     D  M1_Delvr_sts                 10U 0
     D  M1_Prio_sts                  10U 0
```

The status array for a queue property structure is analogous, for example:

```
      * aStatus array of queue-property status codes
     D Q_Status        DS                  INZ
     D  Q_Path_sts                   10U 0
     D  Q_Label_sts                  10U 0
     D  Q_Xact_sts                   10U 0
```

Data structures (dynamic method)

You can create a property structure either *statically* or *dynamically* in an RPG program. This section explains the dynamic method, which lets you create a single structure containing a varying set of message, queue, or queue manager properties.

✓ *For a complete explanation of the data structures, see* Data structures (static method) *on page 30.*

Dynamic method

Suppose that your application creates a queue and sends and receives messages containing varying sets of message properties. Before you call the `MQCreateQueue` API function, you need to create a queue property structure including several queue properties. Before you call `MQSendMessage` and `MQRcvMessage`, you need to create a message property structure containing a variable number of message properties.

Using the static method, you would need to define a separate property structure for each set of message or queue properties that the program needs. Using the dynamic method, you can define a single property structure that accommodates all the combinations.

In an RPG program, you can implement a dynamic property structure using arrays or multiple-occurrence data structures. In the definition specifications, you need to define the maximum size of the arrays or the maximum number of occurrences. You also need to define a base pointer to the first element or occurrence.

In the calculation specifications, the program sets the number of active elements or occurrences, that is, the number of properties included in the structure. The program then moves the desired queue or message properties into the arrays or structures.

In this way, the program can change the set of properties before each Envoy MQ Client API call.

Property structure

At the top level, a dynamic property structure definition is similar to a static definition. The differences are the following:

- You need to define only one property structure, where you can afterwards store queue or message properties as needed.
- You must give a name to the first field in the property structure (`cProp` in the example). Before each API call, you must assign a value to this field indicating how many properties are actually in the property structure.

```
      * Top-level property structure
     D Props           DS
     D  cProp                        10U 0 INZ(0)
     D  aPropID                        *   INZ(%ADDR(MQ_PropID))
     D  aPropVar                       *   INZ(%ADDR(PROPVRIANT))
     D  aStatus                        *   INZ(%ADDR(MQ_Result))
```

Substructures

In the dynamic method, define the substructures of a property structure as RPG arrays or multiple-occurrence data structures. Set the array size or the number of occurrences to the maximum number of properties in any single API call, anywhere in your program.

The following examples illustrate the definitions for a dynamic property structure containing up to 10 properties.

Array of property identifiers

The array of property identifiers corresponds to the aProp field of a property structure in C. In RPG, you can define the array as follows:

```
 * aProp array of up to 10 property identifiers
D MQ_PropID       S             10U 0 DIM(10)
```

Array of propvariant structures

In RPG, the array of propvariant structures is defined as a multiple-occurrence data structure. The structure corresponds to the aPropVar field of a property structure in C.

Note the use of the FMQPROPVAR copy member (which is found in the QRPGLESRC file of the Envoy MQ Client library) as the array element. FMQPROPVAR contains a complete RPG definition of the propvariant data structure.

```
 * aPropVar array of property values
D PROPVRIANT      DS                  OCCURS(10)
D/COPY FMQPROPVAR
```

Array of status codes

The array of status codes corresponds to the aStatus field in C. A sample definition follows:

```
 * aStatus array of property status codes
D MQ_Result       S             10U 0 DIM(10)
```

Using a dynamic property structure

There are several steps to use a dynamic property structure. The following is a typical procedure:

1. In the cProp field of the top-level structure, set the number of properties that you want to include in the structure.
2. Clear the substructures.

3. Move the desired property identifiers (`PID_...` constants) into the `aProp` array.
4. Move the value types and values of each property into the `aPropVar` array.
5. If a property requires a buffer, store the value in the buffer.
6. Call the desired Envoy MQ Client API function providing the property structure as a parameter.

In the following example, we send a message including two message properties:

❑ Message body
❑ Message label

✓ *For a more comprehensive example (including additional properties, receiving a message, and creating a queue), see the FMQRDYN program which is described in* Online samples, *page 48.*

```
D M_Label         S            124A
D hRes            S             10U 0
D Handle          S             10U 0
D Body            S            100A
D pTransaction    S             16A
 *
 *   For simplicity, the steps of obtaining a queue handle and beginning
 *   a transaction are omitted here. See the FMQRDYN sample program for
 *   these steps.
 *
 *   1. Set the number of properties in the property structure
C                   Eval      cProp = 2
 *
 *   2. Clear the substructures
 *
C                   Clear                   MQ_PropID
C                   Clear                   MQ_Result
C                   Clear     *ALL          PROPVRIANT
 *
 *   3. Set the message property identifiers in the aProp array
 *
C                   Move      PID_M_BODY    MQ_PropID(1)
C                   Move      PID_M_LABEL   MQ_PropID(2)
 *
 *   4. Set the property values in the aPropVar array
 *
 *     The field names (vt, cElems, pElems, and pwszVal) are defined
 *         in the FMQPROPVAR copy member
C     1             Occur     PROPVRIANT
C                   Move      VT_VECTOR#UI1 vt
```

```
    C                   Eval      cElems = 50
    C                   Eval      pElems = %ADDR(Body)
    C     2             Occur     PROPVRIANT
    C                   Move      VT_LPWSTR     vt
    C                   Eval      pwszVal = %ADDR(M_Label)
     *
     *   5. Set the buffers for the property values
     *
    C                   Do        5             I                 1 0
    C                   MoveL     I             cI                1
     *
     *         The message body is 'Message number <loop index>',
     *         padded with blanks up to the value of cElems (50)
    C                   Eval      Body = 'Message number ' + cI
     *
     *         The message label is 'Label number <loop index>',
     *         converted to a null-terminated string
    C     2             Occur     PROPVRIANT
    C                   Eval      %str(pwszVal:50) = 'Label number ' +  cI
     *
     *   6. Call the send-message API
     *
    C                   Eval      hRes = MQSendMessage(Handle          :
    C                                                 %ADDR(Props)     :

    C                                                 pTransaction)
    C                   EndDo
```

String handling

Several of the message, queue, and queue manager properties have values that are character strings. For example, the message label (`PID_M_LABEL`) is a string of up to 250 characters. In addition, certain Envoy MQ Client API functions (for example `FMQConnect`), require parameters that are strings.

This section describes the differences between C and RPG strings and the steps to ensure compatibility of your programs with the MSMQ standard.

✓ *For details of the maximum string length, etc., see the Microsoft MSMQ documentation and SDK online help.*

Null-terminated strings

MSMQ and Envoy MQ Client require that every string value be terminated by a null character. In RPG, strings are predefined in length and are padded with trailing blanks. You can convert strings between the two formats using the RPG built-in function `%STR`, for example:

```
     D M_Label         S            124A
     D pM_Label        S               *
     C                   Eval       M_Label =%STR(pM_Label:124)
```

You can also create a null-terminated string by concatenating `X'00'` at the end of the meaningful text, for example:

```
     D M_Label         S            124A
     C                   Eval      M_Label = 'Test message ' + X'00'
```

To make sure that the null character is added to the end of the meaningful text use the built-in function `%TRIM`, for example:

```
     D M_Label       S            124A
     C                 Eval      M_Label = %TRIM('Test message      ') + X'00'
```

EBCDIC to UNICODE conversion

Envoy MQ Client uses a code-page translation table to translate string properties and parameters from EBCDIC to UNICODE or vice versa.

All message and queue properties are converted, with the following exceptions:

- The message body (`PID_M_BODY`) is converted only if the message body type (`PID_M_BODY_TYP`) is `VT_LPWSTR` or `VT_BSTR`. Envoy MQ does not translate a message body of any other type because it doesn't know whether the body contains text or binary data. Instead, you should program whatever conversions are needed.
- The message extension (`PID_M_EXT`).

Sample program

This section presents the complete source code of the `FMQBOOK` sample program, which is supplied online in the `Samples` file of the Envoy MQ Client library. The program illustrates some basic messaging operations, including:

- ❑ Creating and deleting a queue
- ❑ Opening and closing a queue
- ❑ Sending and receiving a message

The program uses the static method to create the required MSMQ and Envoy MQ data structures. For a detailed discussion of the structures, see *Data structures (static method)* on page 30.

✓ *For additional sample programs, see* Online samples *on page 48.*

Source code

```
H
 ****************************************************************
 *                                                              *
 * Program name: FMQBOOK                                        *
 *                                                              *
 * Description:  Sample ILE RPG program demonstrating basic     *
 *               Envoy MQ messaging operations                  *
 *                                                              *
 * Envoy MQ Client for AS/400                                   *
 * (C) Copyright 2002 by Envoy Technologies Inc.                *
 * All rights reserved                                          *
 *                                                              *
 ****************************************************************
 *
 *
 * Include Envoy MQ definitions in the program
D/COPY FMQCONST
 *
 * aProp array of queue property identifiers
D Q_Props         DS
D                               10U 0 INZ(PID_Q_PATH)
D                               10U 0 INZ(PID_Q_LABEL)
D                               10U 0 INZ(PID_Q_XACT)
 *
 * aStatus array of queue-property status codes
D Q_Status        DS                  INZ
D  Q_Path_sts                   10U 0
D  Q_Label_sts                  10U 0
D  Q_Xact_sts                   10U 0
 *
 * aPropVar array of queue property values
D Q_Values        DS
 *
 * Propvariant structure specifying the queue path name
D                                5U 0 INZ(VT_LPWSTR)
D                               14A   INZ(MQ_Reserved)
D  pQPath                         *   INZ(%ADDR(Q_Path))
D                                 *
 *
 * Propvariant structure specifying the queue label
D                                5U 0 INZ(VT_LPWSTR)
D                               14A   INZ(MQ_Reserved)
D  pQLabel                        *   INZ(%ADDR(Q_Label))
D                                 *
 *
 * Propvariant structure specifying the queue transaction status
D                                5U 0 INZ(VT_UI1)
```

```
D                               14A   INZ(MQ_Reserved)
D  Q_Xact                        1A
D                                 *
 *
 * Queue property structure
D Q_QProps        DS
D                               10U 0 INZ(3)
D                                 *   INZ(%ADDR(Q_Props))
D                                 *   INZ(%ADDR(Q_Values))
D                                 *   INZ(%ADDR(Q_Status))
 *
 * aProp array of message property identifiers
D M1_Props        DS
D                               10U 0 INZ(PID_M_BODY)
D                               10U 0 INZ(PID_M_DELIVERY)
D                               10U 0 INZ(PID_M_PRIORITY)
 *
 * aStatus array of message-property status codes
D M1_Status       DS                  INZ
D  M1_Body_sts                  10U 0
D  M1_Delvr_sts                 10U 0
D  M1_Prio_sts                  10U 0
 *
 * aPropVar array of message property values
D M1_Values       DS
 *
 * Propvariant structure specifying the message body
 * (The body is stored in a 100-byte buffer M1_Body)
D                                5U 0 INZ(VT_VECTOR#UI1)
D                               14A   INZ(MQ_Reserved)
D                               10U 0 INZ(100)
D                                 *   INZ(%ADDR(M1_Body))
 *
 * Propvariant structure specifying the message delivery
 * (MQ_DLV_RECVRBL means recoverable delivery, guaranteed even after
 * recovery from a crash)
D                                5U 0 INZ(VT_UI1)
D                               14A   INZ(MQ_Reserved)
D                                2B 0 INZ(MQ_DLV_RECVRBL)
D                                 *
 *
 * Propvariant structure specifying a message priority of 3
D                                5U 0 INZ(VT_UI1)
D                               14A   INZ(MQ_Reserved)
D                                2B 0 INZ(3)
D                                 *
 *
 * Message property structure
D M1_MsgProps     DS
D  M1_cProp                     10U 0 INZ(3)
D  M1_aPropID                     *   INZ(%ADDR(M1_Props))
D  M1_aPropVar                    *   INZ(%ADDR(M1_Values))
```

```
D  M1_aStatus                     *   INZ(%ADDR(M1_Status))
 *
 * Standalone field definitions
 *
 * Buffers for the queue path and label
D Q_Path          S             50A
D Q_Label         S            100A
 * Return code of Envoy MQ Client API functions
D hRes            S             10U 0
 * Envoy MQ Connector connection handle
D hConn           S             10U 0
 * Queue handle
D Q1_Handle       S             10U 0
 * Queue format name buffer
D Q1_FmtName      S            256A
 * Length of the queue format name buffer
D Q1_FmtNameLng   S             10U 0 INZ(50)
 * Buffer for the message body (initialized with a test message)
D M1_Body         S            100A   INZ('Hello, world')
 *
 *--------------------------------------------------------------------*
 *
 * Create a queue called '.\AS400SAMPLE'
C                   EVAL      %str(pQPath : 50) = '.\AS400SAMPLE'
C                   EVAL      %str(pQLabel: 100) = 'AS400 Test Queue'
C                   EVAL      Q_Xact = MQ_Q_XACT_NONE
C                   EVAL      hRes = MQCreateQueue(MQ_ACCESS_ALL    :
C                                                  %ADDR(Q_QProps)  :
C                                                   Q1_FmtName      :
C                                                   Q1_FmtNameLng)
 *
 * Open the queue for sending
C                   EVAL      hRes = MQOpenQueue(Q1_FmtName    :
C                                                MQ_SEND       :
C                                                MQ_DENY_NONE  :
C                                                Q1_Handle)
 *
 * Send a message
C                   EVAL      hRes = MQSendMessage(Q1_Handle          :
C                                                  %ADDR(M1_MsgProps) :
C                                                  MQ_NO_XACT)
 *
 * Close the queue
C                   EVAL      hRes = MQCloseQueue(Q1_Handle)
 *
 * Open the queue for receiving
C                   EVAL      hRes = MQOpenQueue(Q1_FmtName    :
C                                                MQ_RECEIVE    :
C                                                MQ_DENY_SHARE :
C                                                Q1_Handle)
 *
 * Receive the message
```

```
     C                   MOVE      *BLANKS       M1_Body
     C                   EVAL      hRes = MQReceiveMessage(Q1_Handle     :
     C                                                MQ_INFINITE        :
     C                                                MQ_ACT_RECEIVE     :
     C                                                %ADDR(M1_MsgProps) :
     C                                                *NULL              :
     C                                                *NULL              :
     C                                                0                  :
     C                                                MQ_NO_XACT)
      *
      * Display the message body
     C                   EVAL      Res = %subst(M1_Body:1:50)
     C                   DSPLY                   Res              50
      *
      * Close the queue
     C                   EVAL      hRes = MQCloseQueue(Q1_Handle)
      *
      * Delete the queue
     C                   EVAL      hRes = MQDeleteQueue(Q1_FmtName)
      *
      * Disconnect from Envoy MQ Connector
     C                   EVAL      hRes= FMQDisconnect(hConn)
      *
     C                   SETON                                        LR
```

Online samples

The Envoy MQ Client library includes several online programs and source members that you can use in your RPG applications.

File	Description
QRPGLESRC	Copy members that you can include in your applications
SAMPLES	Sample programs illustrating various Envoy MQ programming techniques and solutions to programming problems

The following paragraphs describe the online samples in more detail.

Copy members

FmqConst You should include the `FMQCONST` copy member in every Envoy MQ Client RPG application.

This member contains definitions of MSMQ properties, named constants, and API functions. For a complete description, see *FMQCONST copy member* on page 19.

FmqPropvar The `FMQPROPVAR` copy member provides a complete RPG definition of the MSMQ propvariant data structure. For an explanation of the propvariant structure, see *Substructures of property structures* on page 33.

The member is recommended for use in programs that create property structures dynamically. For an example of its use, see the `FMQRDYN` sample program.

FmqLocate The `FMQLOCATE` copy member defines the data structures used in queue queries.

The member is recommended for use in programs that create the query structures dynamically. For an example, see the `FMQRDYNLOC` sample program.

Sample programs

FmqBook `FMQBOOK` is a sample program illustrating basic messaging operations.

The program provides example of:

- ❑ Creating and deleting a queue
- ❑ Opening and closing a queue
- ❑ Sending and receiving a message

The complete source code of this program is printed in the *Sample program* section of this chapter on page 44.

FmqrStc `FMQRSTC` is a sample program illustrating messaging operations.

The program provides examples of:

- ❑ Connecting to and disconnecting from Envoy MQ Connector
- ❑ Creating and deleting a queue
- ❑ Converting a queue path name to a format name
- ❑ Opening and closing a queue
- ❑ Sending and receiving a message

FmqrXact

FMQRXACT is a sample program illustrating transacted messaging.

The program sends and receives a set of messages in a single MSMQ transaction. The program provides examples of:

❑ Connecting to and disconnecting from Envoy MQ Connector
❑ Creating a transactional queue
❑ Opening, and closing a queue
❑ Sending a set of messages
❑ Beginning and committing an MSMQ transaction
❑ Receiving a set of messages in a single MSMQ transaction

FmqrDyn

FMQRDYN is a sample program illustrating the dynamic creation of property structures. The program uses the FMQPROPVAR copy member to define the propvariant data structure.

The program illustrates most of the common messaging operations, such as:

❑ Creating and opening a queue
❑ Sending and receiving an authenticated message

- ❑ Sending and receiving transacted messages
- ❑ Disconnecting from Envoy MQ Connector

FmqrLoc

FMQRLOC is a sample program illustrating how to construct a queue query. The program calls the MQLocateBegin, MQLocateNext, and MQLocateEnd functions to find a queue having a specified label.

FmqrDynLoc

FMQRDYNLOC is a sample program that creates a queue query dynamically. The program illustrates the use of the FMQLOCATE copy member, and finds a queue having a specified label.

FmqrLog

FMQRLOG illustrates the Envoy MQ Client debug logging functions (FMQDebug, FMQGetLogPath, and FMQSetLogPath).

FmqrVer

FMQRVER displays Envoy MQ Client version information on the screen. It illustrates the use of the FMQVersion API function.

Chapter 4

COBOL Interface

The Envoy MQ Client for AS/400 provides a COBOL interface, which lets you call the Envoy MQ Client API functions directly from your COBOL programs. The interface provides all the needed COBOL definitions, so you can access the complete API without any C programming at all.

The COBOL interface is similar to the RPG interfaces described in chapters 3 and 4 of this book. If you are already familiar with one of the RPG interfaces, you will find the COBOL interface very easy to learn.

Operating system requirements

The COBOL interface described in this chapter runs on OS/400 version V3R2 or higher.

Overview of the interface

The interface is implemented as a set of external API procedures and copy members. This chapter explains:

- ❑ The steps for creating a Envoy MQ Client application in COBOL
- ❑ The structure and contents of `FMQCONST`, which is the most important of the copy members
- ❑ Techniques for calling the Envoy MQ Client API procedures

The interface provides two additional copy members, called `FMQPROPVAR` and `FMQLOCATE`, which support dynamic programming techniques for building message and queue property structures. The chapter includes:

- ❑ Sample COBOL data structures representing MSMQ message and queue properties, constructed using the dynamic techniques
- ❑ Sample COBOL messaging applications

API functions

This chapter describes an interface that you can use to call the Envoy MQ Client API functions in COBOL programs. It does not document the API functions themselves. For information on that subject, see *API implementation* on page 13 and the references therein.

Programming steps

✓ *You can run COBOL applications with the Envoy MQ Client version for OS/400 V3R2 or the version for OS/400 V3R7. The COBOL interfaces of the two versions are identical.*

To program a Envoy MQ Client messaging application, follow these steps:

1. Copy the `FMQCONST` member, which is found in the `QCBLLESRC` file of the Envoy MQ Client library, into the working storage section of your COBOL program (see *FMQCONST copy member* on page 55).
2. Optionally, copy the `FMQPROPVAR` and/or `FMQLOCATE` members into the working storage section of your program. These members can help you set up the data structures you need for Envoy MQ Client API calls (see *Data structures* on page 65).
3. Create COBOL definitions for the required message and queue properties (see *Data structures* on page 65).
4. Code the Envoy MQ Client API calls (see *Sample program* on page 74).
5. Compile the program to an ILE module using the IBM ILE COBOL/400 compiler.

For example, if your module should be called `YOURLIB/YOURMOD`, issue the following commands:

```
ADDLIBLE ENVOY MQ
CRTCBLMOD MODULE(YOURLIB/YOURMOD)
SRCFILE(YOURLIB/YOURFILE) +
SRCMBR(YOURMEMBER) OPTION(*NOMONOPRC *APOST)
```

6. Bind the ILE module to the following Envoy MQ ILE service program:

```
FALCONMQ/FMQDCLIB
```

For example:

```
CRTPGM PGM(YOURLIB/YOURMOD) +
MODULE(YOURLIB/YOURMOD) BNDSRVPGM(FALCONMQ/FMQDCLIB)
ACTGRP(*NEW)
```

FMQCONST copy member

The `FMQCONST` copy member provides the definitions that you need to access the Envoy MQ Client API. You must copy `FMQCONST` into the working storage section of your COBOL program. `FMQCONST` is found in the `QCBLLESRC` file of the Envoy MQ Client library.

The `FMQCONST` definitions include:

- ❑ Constants representing message properties
- ❑ Constants representing queue properties
- ❑ Constants representing queue manager properties
- ❑ Constants representing the value types of properties
- ❑ Miscellaneous named constants

In general, the definitions are very similar to the C-language definitions in the C header files, `mq.h`, `wintypes.h`, and `fmqpubd.h`, which are also supplied with Envoy MQ Client. The main difference is that the COBOL identifiers contain hyphens (-) rather than underscores (_). For example, the C constant `PROPID_M_DEST_QUEUE_LEN` (representing the message property *destination queue name length*) is represented as `PROPID-M-DEST-QUEUE-LEN` in COBOL.

Message properties

The following table lists the constants representing message properties in COBOL. The constants are identical to the property identifiers in C, except that underscores (_) are replaced with hyphens (-).

The table also lists the following information, which is needed to construct a propvariant structure for each property (see *Substructures of property structures* on page 68):

- ❑ The value type constant of the property in COBOL (for a list of the corresponding constants in C, see *Value type constants* on page 59)
- ❑ The data type of the property value

✓ *The* VT-NULL *value types are permitted only when receiving a message. See the Microsoft MSMQ documentation for complete details about the meaning of each property and the permitted values and types.*

Message property	Value type	Data type
PROPID-M-ACKNOWLEDGE	VT-UI1 (or VT-NULL)	PIC X
PROPID-M-ADMIN-QUEUE	VT-LPWSTR	POINTER
PROPID-M-ADMIN-QUEUE-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-APPSPECIFIC	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-ARRIVEDTIME	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-AUTH-LEVEL	VT-UI4	PIC 9(9) BINARY
PROPID-M-AUTHENTICATED	VT-UI1 (or VT-NULL)	PIC X
PROPID-M-BODY	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY POINTER
PROPID-M-BODY-SIZE	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-BODY-TYPE	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-CLASS	VT-UI2 (or VT-NULL)	PIC 9(4) BINARY
PROPID-M-CONNECTOR-TYPE	VT-CLSID	POINTER
PROPID-M-CORRELATIONID	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY POINTER
PROPID-M-DELIVERY	VT-UI1 (or VT-NULL)	PIC X
PROPID-M-DEST-QUEUE	VT-LPWSTR	POINTER
PROPID-M-DEST-QUEUE-LEN	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-DEST-SYMM-KEY	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY POINTER
PROPID-M-DEST-SYMM-KEY-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-ENCRYPTION-ALG	VT-UI4	PIC 9(9) BINARY

Message property	Value type	Data type
PROPID-M-EXTENSION	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY POINTER
PROPID-M-EXTENSION-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-HASH-ALG	VT-UI4	PIC 9(9) BINARY
PROPID-M-JOURNAL	VT-UI1	PIC X
PROPID-M-LABEL	VT-LPWSTR	POINTER
PROPID-M-LABEL-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-MSGID	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY POINTER
PROPID-M-PRIORITY	VT-UI1 (or VT-NULL)	PIC X
PROPID-M-PRIV-LEVEL	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-PROV-NAME	VT-LPWSTR	POINTER
PROPID-M-PROV-NAME-LEN	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-PROV-TYPE	VT-UI4	PIC 9(9) BINARY
PROPID-M-RESP-QUEUE	VT-LPWSTR	POINTER
PROPID-M-RESP-QUEUE-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-SECURITY-CONTEXT	VT-UI4	PIC 9(9) BINARY
PROPID-M-SENDER-CERT	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY POINTER
PROPID-M-SENDER-CERT-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-SENDERID	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY POINTER
PROPID-M-SENDERID-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-SENDERID-TYPE	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-SENTTIME	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY

Message property	Value type	Data type
PROPID-M-SIGNATURE	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY POINTER
PROPID-M-SIGNATURE-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-SRC-MACHINE-ID	VT-CLSID	POINTER
PROPID-M-TIME-TO-BE-RECEIVED	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-TIME-TO-REACH-QUEUE	VT-UI4	PIC 9(9) BINARY
PROPID-M-TRACE	VT-UI1 (or VT-NULL)	PIC X
PROPID-M-VERSION	VT-UI4	PIC 9(9) BINARY
PROPID-M-XACT-STATUS-QUEUE	VT-LPWSTR	POINTER
PROPID-M-XACT-STATUS-QUEUE-LEN	VT-UI4	PIC 9(9) BINARY

Queue properties

The following table lists the constants representing queue properties in COBOL.

Queue property	Value type	Data type
PROPID-Q-AUTHENTICATE	VT-UI1	PIC X
PROPID-Q-BASEPRIORITY	VT-I2	PIC S9(4) BINARY
PROPID-Q-CREATE-TIME	VT-I4	PIC S9(9) BINARY
PROPID-Q-INSTANCE	VT-CLSID	POINTER
PROPID-Q-JOURNAL	VT-UI1	PIC X
PROPID-Q-JOURNAL-QUOTA	VT-UI4	PIC 9(9) BINARY
PROPID-Q-LABEL	VT-LPWSTR	POINTER
PROPID-Q-MODIFY-TIME	VT-I4	PIC S9(9) BINARY
PROPID-Q-PATHNAME	VT-LPWSTR	POINTER

Queue property	Value type	Data type
PROPID-Q-PRIV-LEVEL	VT-UI4	PIC 9(9) BINARY
PROPID-Q-QUOTA	VT-UI4	PIC 9(9) BINARY
PROPID-Q-TRANSACTION	VT-UI1	PIC X
PROPID-Q-TYPE	VT-CLSID	POINTER

Queue manager properties

The following table lists the constants representing queue manager properties in COBOL.

Queue manager property	Value type	Data type
PROPID-QM-CONNECTION	VT-VECTOR-LPWSTR	Two fields: PIC S9(9) BINARY POINTER
PROPID-QM-ENCRYPTION-PK	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY POINTER
PROPID-QM-MACHINE-ID	VT-CLSID	POINTER
PROPID-QM-PATHNAME	VT-LPWSTR	POINTER
PROPID-QM-SITE-ID	VT-CLSID	POINTER

Value type constants

The following table lists the value type constants defined in FMQCONST and the corresponding constants defined in the C header files. Only the constants that are currently used in MSMQ are listed.

The value types are used in propvariant structures, which store the values of properties. For a full explanation, see *Substructures of property structures* on page 68. For reference, the table also indicates:

- ❑ The data types of the value fields in a propvariant structure
- ❑ The suggested data names for the property values
- ❑ The interpretation of the value fields
- ❑ The names of the corresponding value fields in C

COBOL				**Equivalent in C**		
Value type constant	**Data type**	**Suggested data names**[c]	**Interpretation of property value**	**Value type constant**	**Data type**	**Union field name**
VT-CLSID	POINTER	MQ-PUUID	Base pointer (points to a GUID code, type 16A)	VT_CLSID	CLSID _RPC_FAR	*puuid
VT-I2	PIC S9(4) BINARY	MQ-IVAL	Property value	VT_I2	short	iVal
VT-I4	PIC S9(9) BINARY	MQ-LVAL	Property value	VT_I4	long	lVal
VT-LPWSTR	POINTER	MQ-LPWSTR	Base pointer (points to a null-terminated string)	VT_LPWSTR	LPWSTR	pwszVal
VT-NULL			No value (permitted only when receiving a message)	VT_NULL		
VT-UI1	PIC X	MQ-BVAL	Property value	VT_UI1	UCHAR	bVal
VT-UI2	PIC 9(4) BINARY	MQ-UIVAL	Property value	VT_UI2	USHORT	uiVal
VT-UI4	PIC 9(9) BINARY	MQ-ULVAL	Property value	VT_UI4	ULONG	ulVal

COBOL				Equivalent in C		
Value type constant	**Data type**	**Suggested data names**[c]	**Interpretation of property value**	**Value type constant**	**Data type**	**Union field name**
VT-VECTOR-LPWSTR	Two fields: PIC S9(9) BINARY POINTER	 MQ-CALPWSTR-CELEMS MQ-CALPWSTR-PELEMS	 Length of buffer Base pointer (points to buffer)[a]	VT_VECTOR | VT_LPWSTR	CALPWSTR	calpwstr
VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY POINTER	 MQ-CAUB-CELEMS MQ-CAUB-PELEMS	 Length of buffer Base pointer (points to buffer)[b]	VT_VECTOR | VT_UI1	CAUI1	caub

Notes

a. For the value type VT-VECTOR-LPWSTR, the buffer contains a null-terminated string.

b. For the value type VT-VECTOR-UI1, the buffer may contain various types of binary or text data:

- The message body property (PROPID-M-BODY) has this value type and may contain any data whatsoever.
- Other properties having this value type are restricted to certain types or structures of data. For information about specific properties, see the Microsoft MSMQ documentation and SDK online help.

c. The data names are defined in the FMQPROPVAR copy member. You can replace the MQ– prefix with another prefix when you copy FMQPROPVAR into your program.

Miscellaneous named constants

FMQCONST defines a large number of constants representing special values of API function arguments, error codes, etc. The following are a few examples:

Constant in COBOL	Equivalent in C
MQ-ACCESS-ALL	PSD_SPECIALACCESS_ALL
MQ-ERROR-ACCESS-DENIED	MQ_ERROR_ACCESS_DENIED
MQ-ERROR-BUFFER-OVERFLOW	MQ_ERROR_BUFFER_OVERFLOW
MQ-LE	PRLE

✓ *The constants are too numerous to list here. For a complete listing, please refer to the `FMQCONST` source code.*

API functions

The COBOL interface provides a complete set of definitions for the Envoy MQ Client API functions. The functions are called as external procedures in COBOL.

Calling syntax

In the procedure section of your program, you can call the `MQSendMessage` procedure using syntax such as the following. The procedure is equivalent to the `MQSendMessage()` function in the MSMQ or Envoy MQ Client API.

```
CALL LINKAGE TYPE IS PROCEDURE 'MQSendMessage'
                     USING BY VALUE      Queue-Handle
                           BY REFERENCE  Props
                           BY VALUE      pTransaction
                     RETURNING MQ-Result-Long.
EVALUATE MQ-Result
    WHEN MQ-OK   GO TO Send-Message-Exit
    WHEN OTHER   DISPLAY ERR-MSG
                 PERFORM Envoy MQ -Disconnect
END-EVALUATE.
```

The procedure accepts three parameters:

`Queue-Handle`	Specifies the destination queue.
`Props`	A message property structure, containing the content of the message.

`pTransaction` A transaction handle of type A(16), specifying a transaction to which the message belongs (optionally `NULL`).

The procedure returns a numerical result code `MQ-Result-Long`.

Comparison with C

For comparison, the following is the corresponding API function declaration in C:

```
HRESULT APIENTRY MQSendMessage(
  QUEUEHANDLE hDestinationQueue,
  MQMSGPROPS * pMessageProps,
  ITransaction * pTransaction
);
```

Samples of other API calls

For other examples of COBOL API calls, see the *Sample program* on page 74.

For a complete set of examples for the Envoy MQ Client API procedures, see the *Online samples* listed on page 84. In the online samples, you can find examples of all the Envoy MQ Client API procedures including:

- ❑ Setting up the input parameters of each procedure
- ❑ The correct syntax for the procedure call
- ❑ Interpreting the output parameters and return values

List of Envoy MQ Client procedures

The following is a list of Envoy MQ Client API procedures. The table includes:

- ❑ The COBOL procedure names, which are identical to the C function names
- ❑ The Envoy MQ Client sample programs where the API calls are illustrated (see *Online samples* on page 84)
- ❑ References for additional information, including a complete explanation of each procedure and its parameters.

The key for the additional references is as follows:

A. The chapter on *Programming Messaging Applications* in the *Envoy MQ Programmer's Guide*.

B. The Microsoft MSMQ documentation and SDK online help

Procedure	Sample programs where illustrated	Additional references
FMQAbort		A

Procedure	Sample programs where illustrated	Additional references
FMQCommit	FMQBDYN	A
FMQConnect	FMQBSTC	A
FMQDebug		A
FMQDisconnect	FMQBDYN, FMQBSTC, FMQBDYNLOC	A
FMQGetLogPath		A
FMQSetLogPath		A
FMQVersion		A
FMQV1Connect		A
MQBeginTransaction	FMQBDYN	A
MQCloseCursor	FMQBSAMPLE	B
MQCloseQueue	FMQBDYN, FMQBSTC	B
MQCreateCursor	FMQBSAMPLE	B
MQCreateQueue	FMQBDYN, FMQBSTC	A, B
MQDeleteQueue	FMQBSTC	B
MQFreeMemory	FMQBDYNLOC	B
MQFreeSecurityContext	FMQBDYN	A, B
MQGetMachineProperties	FMQBSAMPLE	B
MQGetQueueProperties	FMQBSAMPLE	B
MQGetSecurityContext	FMQBDYN	A, B
MQHandleToFormatName	FMQBSAMPLE	B
MQInstanceToFormatName	FMQBSAMPLE	B
MQLocateBegin	FMQBDYNLOC	A, B
MQLocateEnd	FMQBDYNLOC	B
MQLocateNext	FMQBDYNLOC	B
MQOpenQueue	FMQBDYN, FMQBSTC	B
MQPathNameToFormatName	FMQBDYN, FMQBSTC	B

Procedure	Sample programs where illustrated	Additional references
MQReceiveMessage	FMQBDYN, FMQBSTC	A, B
MQRegisterCertificate		A, B
MQSendMessage	FMQBDYN, FMQBSTC	B
MQSetQueueProperties	FMQBSAMPLE	B

Data structures

Many of the MSMQ and Envoy MQ Client API functions require parameters that are pointers to data structures. These include:

Property structures Structures containing sets of message, queue, or queue manager properties. The content of a message, for example, is specified in a message property structure.

Substructures of property structures Structures and arrays that are elements of property structures. An example is the *propvariant structure*, which contains the values of properties.

Query structures Structures required as parameters of the MQLocateBegin function, which searches for queues having specified property values.

This section explains how you can create the property structures and substructures in your COBOL programs. If you wish, you can copy the examples (with minor modifications) into your COBOL programs. You can find additional examples in the *Sample program* on page 74.

For additional information on the interpretation and use of the structures, please refer to the Microsoft MSMQ documentation and SDK online help.

For information on the query structures, please see the *Online samples* described on page 84.

Comparison of RPG and COBOL programming methods

The method described in this section corresponds to the *dynamic method* described for the Envoy MQ Client RPG Interface (see *Data structures (dynamic method)* in Chapter 3, page 38). The dynamic method lets you create a single structure containing a varying set of message, queue, or queue manager properties..

You can also use a *static method* to construct the data structures, but this is less convenient in COBOL than in RPG (see *Data structures (static method)* in Chapter 3, page 30). For an example using static COBOL structures, see the `FMQBSTC` sample program supplied with Envoy MQ Client (see *Online samples* on page 84).

Programming method

Suppose that your application creates a queue and sends and receives messages containing various sets of message properties. Before you call the `MQCreateQueue` API function, you need to create a queue property structure including several queue properties. Before you call `MQSendMessage` and `MQReceiveMessage`, you need to create a message property structure containing the message properties.

In a COBOL program, you can implement the property structure using arrays or multiple-occurrence data structures. In the definition specifications, you need to define the maximum size of the arrays or the maximum number of occurrences. You also need to define pointers to the first element or occurrence.

In the procedure division, the program sets the number of active array elements or occurrences, that is, the number of properties included in the structure. The program then moves the desired queue or message properties into the arrays or structures.

In this way, the program can change the set of properties before each Envoy MQ Client API call.

Property structure

A *property structure* contains a collection of properties and their values. There are three types of property structures, which have different C data types.

Structure	Contains a collection of	C data type
Message property structure	Message properties	MQMSGPROPS
Queue property structure	Queue properties	MQQUEUEPROPS
Queue manager property structure	Queue manager properties	MQQMPROPS

Each property structure contains the following four fields:

COBOL data type	C data type	Field name in C	Description
PIC 9(9) BINARY	DWORD	cProp	A count of the properties included in the structure. The value of this field is the size of the arrays in the other fields of the structure.
POINTER	Array of PROPID	aPropID	A pointer to an array of PROPID_... constants, identifying the properties that are included in the structure (input to the API functions).
POINTER	Array of PROPVARIANT	aPropVar	A pointer to an array of propvariant structures, which contain the values of the properties (input or output).
POINTER	Array of HRESULT	aStatus	A pointer to an array of status codes (output from the API functions).

✓ *In the following discussion, we refer to the fields by their generic names `cProp`, `aPropID`, etc. In COBOL, you must use field names that are unique throughout the entire program.*

The three types of property structures all contain the same four fields. This means that you can represent them in COBOL by defining a single top-level property structure. To create a message property structure, you can store pointers to arrays of message properties in the fields. To create a queue or queue-manager property structure, you can store pointers to arrays of queue properties or queue-manager properties in the fields.

The following is a sample definition of the property structure:

```
* Top level property structure
 01 Props.
    02 cProp         PIC 9(9) BINARY.
    02 aPropID       USAGE IS POINTER.
    02 aPropVar      USAGE IS POINTER.
    02 aStatus       USAGE IS POINTER.
```

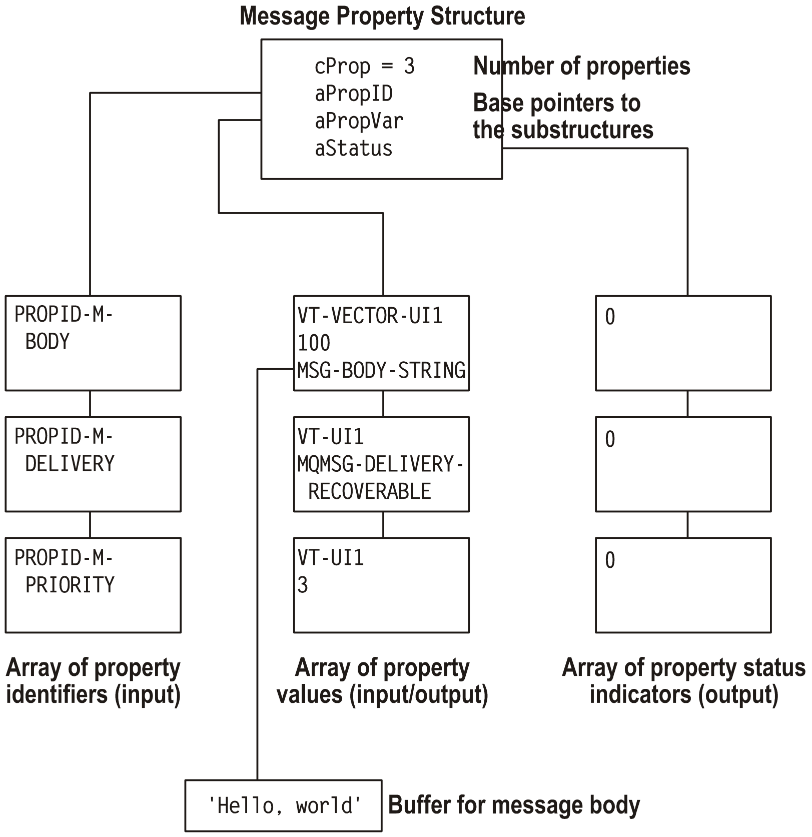
Substructures of property structures

The property structure contains pointers to three arrays:

`aPropID` Pointer to an array of property identifies (`PROPID-...` constants) identifying message, queue, or queue manger properties.

`aPropVar` Pointer to an array of propvariant structures, which contain the values of the properties.

`aStatus` Pointer to an array of status codes, used for output from the API functions.


Message Property Structure
cProp = 3
aPropID
aPropVar
aStatus
Number of properties
Base pointers to the substructures
PROPID-M-BODY
PROPID-M-DELIVERY
PROPID-M-PRIORITY
VT-VECTOR-UI1
100
MSG-BODY-STRING
VT-UI1
MQMSG-DELIVERY-RECOVERABLE
VT-UI1
3
0
0
0
Array of property identifiers (input)
Array of property values (input/output)
Array of property status indicators (output)
'Hello, world'
Buffer for message body

The number of elements in each array is given by the `cProp` field of the property structure. The order of properties must be identical in each array. For example, if the `aPropID` array contains `PROPID_...` constants for the message body, delivery, and priority properties, then the other arrays must also contain elements for exactly the same properties, in the same order.

The following example illustrates how you can construct the arrays in a COBOL program. For convenience, the arrays are represented as multiple-occurrence data structures (in essence, substructures of a property structure) instead of true COBOL arrays.

The example is for a message property structure containing a maximum of 10 properties. We will use the property structure to construct a message containing three properties:

❑ Message body
❑ Message delivery
❑ Message priority

The other seven properties in the property structure are not used in this example.

Setting the number of active properties

The number of properties in the property structure is stored in the cProp field of the property structure. In the sample message, there are three properties. You can specify this in the procedure division by writing:

```
 MOVE 3 TO cProp.
```

This instructs Envoy MQ Client to use the first three properties of the property structure. If any additional properties exist in the structure, they are ignored.

If you later need a property structure containing a different number of properties, you can reset cProp to the new value, up to the array size of the property structure.

Array of property identifiers

The array of property identifiers corresponds to the aProp field of a property structure in C. In COBOL, you can define the array as follows:

```
* aPropID array of up to 10 property identifiers
 01 MQ-PropID-Array.
    02 MQ-PropID PIC 9(9) BINARY OCCURS 10.
```

Here, we have defined the array size for a maximum of 10 properties. Only three of the properties are used in the message example.

In the procedure division, we need to:

❑ Set the aPropID pointer of the property structure to point to the array

❑ Move the property identifiers to the array

For our sample message, we would write:

```
* Set the aPropID pointer of the property structure
 SET aPropID TO ADDRESS OF MQ-PropID-Array.
*
* Move the property identifiers to the array
 MOVE PROPID-M-BODY              TO MQ-PropID(1).
 MOVE PROPID-M-DELIVERY          TO MQ-PropID(2).
 MOVE PROPID-M-PRIORITY          TO MQ-PropID(3).
```

Array of propvariant structures

MSMQ and Envoy MQ Client use propvariant structures to store the values of message, queue, and queue manager properties. On the AS/400, a propvariant is a 48-byte structure containing the following fields:

Value type constant	A `VT-...` constant indicating the data type of the property value.
Reserved	Reserved for future use.
Value1	The value of the property. For certain properties, *Value1* is the size of the value in bytes (equivalent to the `cElems` field in C).
Value2	If *Value1* contains the value, *Value2* is an empty placeholder field. If *Value1* contains the size of the value, then *Value2* is a pointer to the value (equivalent to the `pElems` field in C).

In COBOL, you can define the array of propvariant structures as a multiple-occurrence data structure. The elements of the structure are copies of the `FMQPROPVAR` member, which is supplied in the `QCBLLESRC` file of the Envoy MQ Client library. `FMQPROPVAR` contains a complete COBOL definition of the propvariant data structure.

```
* aPropVar array of up to 10 property values
 01 MQ-PropVar-Array.
    02 MQ-PropVar OCCURS 10.
       COPY FMQPROPVAR REPLACING ==:MQ:== BY ==MQ==.
```

✓ *You can define more than one `aPropVar` array using the `FMQPROPVAR` copy member. In each array, copy `FMQPROPVAR` replacing `:MQ:` with a different string, such as `MQ1`, `MQ2`, etc.*

In the procedure division, we need to:

- Set the `aPropVar` pointer of the property structure to point to the array
- Move the appropriate value type constant, *Value1*, and *Value2* for each message property, to the first three elements of the array

The *Value1* and *Value2* fields in `FMQPROPVAR` have different names and data types depending on the property that you want to store. The names are illustrated in the sample code below. For a complete listing of the *Value* names, see the table of *Value type constants* on page 59.

```
* Set the aPropVar pointer of the property structure
 SET aPropVar TO ADDRESS OF MQ-PropVar-Array.
*
* Set the message body to a 'Hello, World' test string
     MOVE VT-VECTOR-UI1              TO MQ-VARTYPE(1).
* Value1 of the message body property is the length of
the body
     MOVE 12                         TO MQ-CAUB-
CELEMS(1).
* Value2 is a pointer to a buffer containing the message
body
     SET MQ-CAUB-PELEMS(1) TO ADDRESS OF MSG-BODY-
STRING.
*
* Set the delivery property to recoverable
     MOVE VT-UI1                     TO MQ-VARTYPE(2).
* Value1 of the delivery property (there is no Value2)
     MOVE MQMSG-DELIVERY-RECOVERABLE TO MQ-BVAL(2).
*
* Set the priority property to a value of 3
     MOVE VT-UI1                     TO MQ-VARTYPE(3).
* Value1 of the priority property (there is no Value2)
     MOVE 3                          TO MQ-BVAL(3).
```

Elsewhere in the program, you need to define a buffer and store the message in body in it, for example:

```
* Buffer containing a test message body
 77 MSG-BODY-STRING    PIC X(50) VALUE 'Hello, world'.
```

Array of status codes

The array of status codes corresponds to the aStatus field in C. A sample definition follows:

```
       01 MQ-Prop-Result-Array.
          02 MQ-Prop-Result PIC 9(9) BINARY OCCURS 10.
```

The status codes are output from various API functions. In the procedure division, you need to set the aStatus pointer in the property structure to the address of the array:

```
 SET aStatus TO ADDRESS OF MQ-Prop-Result-Array.
```

String handling

Several of the message, queue, and queue manager properties have values that are character strings. For example, the message label is a string of up to 250 characters. In addition, certain Envoy MQ Client API functions (for example FMQConnect), require parameters that are strings.

This section describes the differences between C and COBOL strings and the steps to ensure compatibility of your programs with the MSMQ standard.

✓ *For details of the maximum string length, etc., see the Microsoft MSMQ documentation and SDK online help.*

Null-terminated strings

MSMQ and Envoy MQ Client require that every string value be terminated by a null character. In COBOL, strings are predefined in length and are padded with trailing blanks. You can convert strings between the two formats using the COBOL built-in function `STRING`.

EBCDIC to UNICODE conversion

Envoy MQ Client uses a code-page translation table to translate string properties and parameters from EBCDIC to UNICODE or vice versa.

All message and queue properties are converted, with the following exceptions:

- The message body (`PROPID-M-BODY`) is converted only if the message body type (`PROPID-M-BODY-TYPE`) is `VT_LPWSTR` or `VT_BSTR`. Envoy MQ does not translate a message body of any other type because it doesn't know whether the body contains text or binary data. Instead, you should program whatever conversions are needed.
- The message extension (`PROPID-M-EXTENSION`).

Sample program

This section presents the complete source code of the `FMQBDYN` sample program, which is supplied online in the `SAMPLES` file of the Envoy MQ Client library. The program illustrates some basic messaging operations, including:

- Connecting to and disconnecting from Envoy MQ Connector
- Creating and deleting a queue
- Converting a queue path name to a format name
- Opening and closing a queue
- Sending and receiving a message

The program uses the dynamic method to create the required MSMQ and Envoy MQ Client data structures. For a detailed discussion of the structures, see *Data structures* on page 65.

✓ *For additional sample programs, see* Online samples *on page 84.*

Source code

```
 IDENTIFICATION DIVISION.
    PROGRAM-ID. FMQBDYN.
****************************************************************
*                                                              *
* Description: Sample ILE COBOL/400 program demonstrating the  *
*              use of dynamic property structures and the      *
*              FMQCONST and FMQPROPVAR copy members            *
*                                                              *
* Ver:         V3R2                                            *
*                                                              *
* Envoy MQ Client for AS/400                                   *
* (C) Copyright 2002 by Envoy Technologies Inc.                  *
* All rights reserved                                          *
*                                                              *
****************************************************************
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
* Include Envoy MQ definitions in the program
     COPY FMQCONST OF QCBLLESRC.
*
* aPropID array of up to 10 property identifiers
 01 MQ-PropID-Array.
    02 MQ-PropID PIC 9(9) BINARY OCCURS 10.
*
* aPropVar array of up to 10 property values
* Note : This sample uses the same property structure for both
*        Queue and Message properties. You may define additional
*        property structures using the COPY REPLACING feature.
 01 MQ-PropVar-Array.
    02 MQ-PropVar OCCURS 10.
       COPY FMQPROPVAR REPLACING ==:MQ:== BY ==MQ==.
*
```

```
* aStatus array of up to 10 property status codes
 01 MQ-Prop-Result-Array.
    02 MQ-Prop-Result PIC 9(9) BINARY OCCURS 10.
*
* Top level property structure
 01 Props.
    02 cProp         PIC 9(9) BINARY.
    02 aPropID       USAGE IS POINTER.
    02 aPropVar      USAGE IS POINTER.
    02 aStatus       USAGE IS POINTER.
*
 77 MQ-Result      PIC X(4).
 77 MQ-Result-Long REDEFINES MQ-Result PIC 9(9) BINARY.
*
 77 FormatName         PIC  X(125).
 77 FormatName-Length  PIC S9(9) BINARY.
 77 Queue-Handle       PIC 9(9) BINARY.
 77 Connection-Handle  PIC 9(9) BINARY VALUE 0.
 77 SecContext-Handle  PIC 9(9) BINARY.
 77 pTransaction       USAGE IS POINTER.
 77 Q-PATH-STRING      PIC X(125).
 77 Q-LABEL-STRING     PIC X(125).
 77 MSG-COUNTER        PIC 9(3).
 77 MSG-BODY-STRING    PIC X(50).
 77 MSG-BODY-PREFIX    PIC X(15) VALUE 'Message number '.
 77 MSG-LABEL-STRING   PIC X(22).
 77 MSG-LABEL-PREFIX PIC X(14) VALUE 'Message label '.
 77 ERR-MSG      PIC X(23) VALUE 'Envoy MQ call failed!'.
 77 AUTH-Msg     PIC X(30) VALUE 'Authenticated message received'.
 77 NOT-AUTH-Msg PIC X(30) VALUE 'Unauthenticated message! '.
*-----------------------------------------------------------------
 PROCEDURE DIVISION.
*
 Main SECTION.
*
 Main-P.
*
* Set the pointers of the property structure. The same structure is
* used for both queue and message properties.
     SET aPropID  TO ADDRESS OF MQ-PropID-Array.
     SET aPropVar TO ADDRESS OF MQ-PropVar-Array.
     SET aStatus  TO ADDRESS OF MQ-Prop-Result-Array.
*
* Create a queue if it doesn't already exist
     PERFORM Create-Queue.
* Open the queue for sending
     PERFORM Open-Queue-Send.
```

```
* Send 6 transacted, authenticated messages to the queue
     PERFORM Get-Security-Context.
     PERFORM Begin-Transaction.
     PERFORM Send-Message
          VARYING MSG-COUNTER FROM 1 BY 1 UNTIL MSG-COUNTER = 6.
     PERFORM Commit-Transaction.
     PERFORM Free-Security-Context.
* Close the queue
     PERFORM Close-Queue.
*
* Reopen the queue for receiving
     PERFORM Open-Queue-Receive.
* Receive the first message from the queue
     PERFORM Receive-Message.
* Close the queue
     PERFORM Close-Queue.
*
* Disconnect from Envoy MQ Connector
     PERFORM EnvoyMQ-Disconnect.
*-----------------------------------------------------------------
 Create-Queue SECTION.
*
 Create-Queue-P.
*
* Set the parameters for an MQCreateQueue call
* 1. Create a property structure including five queue properties
* 1.1 Set the queue property names in the MQ-PropID array
     MOVE PROPID-Q-PATHNAME      TO MQ-PropID(1).
     MOVE PROPID-Q-LABEL         TO MQ-PropID(2).
     MOVE PROPID-Q-TRANSACTION   TO MQ-PropID(3).
     MOVE PROPID-Q-TYPE          TO MQ-PropID(4).
     MOVE PROPID-Q-BASEPRIORITY  TO MQ-PropID(5).
*
* 1.2 Set the property values in the MQ-PropVar array
     MOVE VT-LPWSTR          TO  MQ-VARTYPE(1).
     SET MQ-LPWSTR(1)        TO  ADDRESS OF  Q-PATH-STRING.
*
     MOVE VT-LPWSTR          TO  MQ-VARTYPE(2).
     SET MQ-LPWSTR(2)        TO  ADDRESS OF  Q-LABEL-STRING.
*
     MOVE VT-UI1             TO MQ-VARTYPE(3).
     MOVE MQ-TRANSACTIONAL   TO MQ-BVAL(3).
*
     MOVE VT-CLSID           TO  MQ-VARTYPE(4).
     SET MQ-PUUID(4)         TO  ADDRESS OF  MQ-QTYPE-TEST.
*
     MOVE VT-I2              TO  MQ-VARTYPE(5).
     MOVE -2                 TO  MQ-IVAL(5).
*
* 1.3 Set the total number of active properties in the property
* structure
     MOVE  5    TO cProp.
*
* 2. Set the queue path name and label
     STRING '.\AS400SAMPLE' LOW-VALUE
                 DELIMITED BY SIZE INTO Q-PATH-STRING.
     STRING 'AS400 Test Queue' LOW-VALUE
                DELIMITED BY SIZE INTO Q-LABEL-STRING.
*
* 3. Assign a buffer for the queue format name (output)
     MOVE LENGTH OF FormatName TO FormatName-Length.
*
* Call the MQCreateQueue API function to create the queue
     CALL LINKAGE TYPE IS PROCEDURE 'MQCreateQueue'
                           USING  BY VALUE MQ-ACCESS-ALL
                                  BY REFERENCE Props
                                               FormatName
                                               FormatName-Length
                           RETURNING MQ-Result-Long.
     EVALUATE MQ-Result
        WHEN MQ-OK  GO TO Create-Queue-Exit
        WHEN MQ-ERROR-QUEUE-EXISTS PERFORM Path-To-FormatName
        WHEN OTHER DISPLAY ERR-MSG
                   PERFORM EnvoyMQ-Disconnect
     END-EVALUATE.
 Create-Queue-Exit.
        EXIT.
*-----------------------------------------------------------------
 Path-To-FormatName SECTION.
*
 Path-To-FormatName-P.
*
* If a queue with the given path name already exists, call
* MQPathNameToFormatName to retrieve its format name
        CALL LINKAGE TYPE IS PROCEDURE 'MQPathNameToFormatName'
                          USING BY REFERENCE Q-PATH-STRING
                                             FormatName
                                             FormatName-Length
                                  RETURNING MQ-Result-Long.
        EVALUATE MQ-Result
            WHEN MQ-OK   GO TO Path-To-FormatName-Exit
            WHEN OTHER   DISPLAY ERR-MSG
                         PERFORM EnvoyMQ-Disconnect
        END-EVALUATE.
 Path-To-FormatName-Exit.
          EXIT.
*-----------------------------------------------------------------
 Open-Queue-Send SECTION.
*
 Open-Queue-Send-P.
*
* Call MQOpenQueue to open the queue for sending
        CALL  LINKAGE TYPE IS PROCEDURE 'MQOpenQueue'
                           USING BY REFERENCE FormatName
                                 BY VALUE     MQ-SEND-ACCESS
                                              MQ-DENY-NONE
                                 BY REFERENCE Queue-Handle
                           RETURNING MQ-Result-Long.
        EVALUATE MQ-Result
            WHEN MQ-OK   GO TO Open-Queue-Send-Exit
            WHEN OTHER   DISPLAY ERR-MSG
                         PERFORM EnvoyMQ-Disconnect
        END-EVALUATE.
   Open-Queue-Send-Exit.
          EXIT.
*-----------------------------------------------------------------
 Open-Queue-Receive SECTION.
*
 Open-Queue-Receive-P.
*
* Call MQOpenQueue to open the queue for receiving
        CALL  LINKAGE TYPE IS PROCEDURE 'MQOpenQueue'
                           USING BY REFERENCE FormatName
                                 BY VALUE  MQ-RECEIVE-ACCESS
                                           MQ-DENY-RECEIVE-SHARE
                                 BY REFERENCE Queue-Handle
                           RETURNING MQ-Result-Long.
        EVALUATE MQ-Result
            WHEN MQ-OK   GO TO Open-Queue-Receive-Exit
            WHEN OTHER   DISPLAY ERR-MSG
                         PERFORM EnvoyMQ-Disconnect
        END-EVALUATE.
   Open-Queue-Receive-Exit.
          EXIT.
*-----------------------------------------------------------------
 Get-Security-Context SECTION.
*
* Retrieve security information needed to authenticate messages
* using an internal (MSMQ) certificate. The certificate must
* be registered for the current user on the Envoy MQ Connector
* machine.
```

```
*
 Get-Security-Context-P.
*
        CALL  LINKAGE TYPE IS PROCEDURE 'MQGetSecurityContext'
                          USING BY VALUE     NULL
                                BY VALUE     0
                                BY REFERENCE SecContext-Handle
                          RETURNING MQ-Result-Long.
        EVALUATE MQ-Result
            WHEN MQ-OK   GO TO Get-Security-Context-Exit
            WHEN OTHER   DISPLAY ERR-MSG
                         PERFORM EnvoyMQ-Disconnect
        END-EVALUATE.
   Get-Security-Context-Exit.
          EXIT.
*-----------------------------------------------------------------
 Free-Security-Context SECTION.
*
 Free-Security-Context-P.
*
        CALL  LINKAGE TYPE IS PROCEDURE 'MQFreeSecurityContext'
                             USING BY VALUE  SecContext-Handle.
 Free-Security-Context-Exit.
          EXIT.
*-----------------------------------------------------------------
 Begin-Transaction SECTION.
*
 Begin-Transaction-P.
*
* Begin a transaction
        CALL  LINKAGE TYPE IS PROCEDURE 'MQBeginTransaction'
                          USING BY REFERENCE pTransaction
                          RETURNING MQ-Result-Long.
        EVALUATE MQ-Result
            WHEN MQ-OK   GO TO Begin-Transaction-Exit
            WHEN OTHER   DISPLAY ERR-MSG
                         PERFORM EnvoyMQ-Disconnect
        END-EVALUATE.
   Begin-Transaction-Exit.
          EXIT.
*-----------------------------------------------------------------
 Send-Message SECTION.
*
 Send-Message-P.
*
* Send a message and ask MSMQ to authenticate it.
*
```

```
* 1. Create a property structure including four message properties
* 1.1 Set the strings for the message body and label properties
*     (The message body is 'Message number <i>'. The message label
*     is 'Message label <i>'.)
        STRING MSG-BODY-PREFIX MSG-COUNTER
                           DELIMITED BY SIZE INTO MSG-BODY-STRING.
        STRING MSG-LABEL-PREFIX MSG-COUNTER LOW-VALUE
                           DELIMITED BY SIZE INTO MSG-LABEL-STRING.
*
* 1.2 Set the total number of active properties in the property
*      structure
        MOVE 4 TO cProp.
*
* 1.3 Set the aPropID array containing the message property
*     identifiers
        MOVE PROPID-M-BODY              TO MQ-PropID(1).
        MOVE PROPID-M-LABEL             TO MQ-PropID(2).
        MOVE PROPID-M-AUTH-LEVEL        TO MQ-PropID(3).
        MOVE PROPID-M-SECURITY-CONTEXT  TO MQ-PropID(4).
*
* 1.4 Set the aPropVar array containing the property values
        MOVE VT-VECTOR-UI1    TO MQ-VARTYPE(1).
        MOVE 50               TO MQ-CAUB-CELEMS(1).
        SET MQ-CAUB-PELEMS(1) TO ADDRESS OF MSG-BODY-STRING.
*
        MOVE VT-LPWSTR        TO MQ-VARTYPE(2).
        SET MQ-LPWSTR(2)      TO ADDRESS OF MSG-LABEL-STRING.
*
        MOVE VT-UI4                     TO  MQ-VARTYPE(3).
        MOVE MQMSG-AUTH-LEVEL-ALWAYS    TO  MQ-ULVAL(3).
*
        MOVE VT-UI4            TO  MQ-VARTYPE(4).
        MOVE SecContext-Handle TO  MQ-ULVAL(4).
*
* Call MQSendMessage to send the message
         CALL LINKAGE TYPE IS PROCEDURE 'MQSendMessage'
                              USING BY VALUE      Queue-Handle
                                    BY REFERENCE  Props
                                    BY VALUE      pTransaction
                              RETURNING MQ-Result-Long.
         EVALUATE MQ-Result
             WHEN MQ-OK   GO TO Send-Message-Exit
             WHEN OTHER   DISPLAY ERR-MSG
                         PERFORM EnvoyMQ-Disconnect
         END-EVALUATE.
 Send-Message-Exit.
          EXIT.
```

```
*-----------------------------------------------------------------
 Receive-Message SECTION.
*
 Receive-Message-P.
*
* Receive a message (not as part of a transaction) and check for
* authentication.
*
* Notes on the property settings:
* 1. The BODY and LABEL message properties are left unchanged
*    from the previous send operation.
*    A successful receive will place the message body into
*    MSG-BODY-STRING and the Message Label into MSG-LABEL-STRING.
*
* 2. The AUTH-LEVEL property used in the send operation is replaced
*    with the AUTHENTICATED property to enable authentication
*    checking.
*
* 3. The SECURITY CONTEXT property used in the send operation is
*    replaced with the LABEL-LEN property, which specifies the size
*    of the LABEL buffer
*
* Set the total number of active properties in the property structure
         MOVE 4 TO cProp.
         MOVE PROPID-M-AUTHENTICATED  TO  MQ-PropID(3).
         MOVE VT-NULL TO MQ-VARTYPE(3).
*
* Set the buffer length for the LABEL output
         MOVE PROPID-M-LABEL-LEN      TO  MQ-PropID(4).
         MOVE VT-UI4  TO MQ-VARTYPE(4).
         MOVE 125     to MQ-ulVal(4).
*
* Receive the message
        CALL LINKAGE TYPE IS PROCEDURE 'MQReceiveMessage'
                     USING BY VALUE Queue-Handle
                                    MQ-INFINITE
                                    MQ-ACTION-RECEIVE
                           BY REFERENCE Props
                           BY VALUE NULL
                                    NULL
                                    0
                                    MQ-NO-TRANSACTION
                      RETURNING MQ-Result-Long.
        EVALUATE MQ-Result
            WHEN MQ-OK   GO TO Authentication-Check
            WHEN OTHER   DISPLAY ERR-MSG
                         PERFORM EnvoyMQ-Disconnect
        END-EVALUATE.
*
* Check for authentication of the message
 Authentication-Check.
        IF MQ-BVAL(3) = X'01'
            DISPLAY AUTH-Msg
         ELSE DISPLAY NOT-AUTH-Msg
              GO TO Receive-Call-Exit.
 Receive-Call-Exit.
        EXIT.
*-----------------------------------------------------------------
 Close-Queue SECTION.
*
 Close-Queue-P.
*
* Close the queue
        CALL LINKAGE TYPE IS PROCEDURE 'MQCloseQueue'
                            USING BY VALUE Queue-Handle
                            RETURNING MQ-Result-Long.
        EVALUATE MQ-Result
            WHEN MQ-OK   GO TO Close-Queue-Exit
            WHEN OTHER   DISPLAY ERR-MSG
                         PERFORM EnvoyMQ-Disconnect
        END-EVALUATE.
  Close-Queue-Exit.
          EXIT.
*-----------------------------------------------------------------
 Commit-Transaction SECTION.
*
 Commit-Transaction-P.
*
* Commit the transaction
        CALL  LINKAGE TYPE IS PROCEDURE 'Commit'
                           USING BY REFERENCE pTransaction
                                 BY VALUE     0
                                              0
                                              0
                           RETURNING MQ-Result-Long.
        EVALUATE MQ-Result
            WHEN MQ-OK   GO TO Commit-Transaction-Exit
            WHEN OTHER   DISPLAY ERR-MSG
                         PERFORM EnvoyMQ-Disconnect
        END-EVALUATE.
   Commit-Transaction-Exit.
          EXIT.
*-----------------------------------------------------------------
 EnvoyMQ-Disconnect SECTION.
```

```
*
 EnvoyMQ-Disconnect-P.
*
* Call FMQDisconnect() to close the session with the Envoy MQ
* Connector.
        CALL LINKAGE TYPE IS PROCEDURE 'FMQDisconnect'
                            USING BY VALUE Connection-Handle.
        STOP RUN.
   EnvoyMQ-Disconnect-Exit.
        EXIT.
```

Online samples

The Envoy MQ Client library includes several online programs and source members that you can use in your COBOL applications.

File	Description
`QCBLLESRC`	Copy members that you can include in your applications
`SAMPLES`	Sample programs illustrating various Envoy MQ programming techniques and solutions to programming problems

The following paragraphs describe the online samples in more detail.

Copy members

The following copy members, which are located in the `QCBLLESRC` file of the Envoy MQ Client library, contain code for use in your applications.

FmqConst You should include the `FMQCONST` copy member in every Envoy MQ Client COBOL application.

This member contains definitions of MSMQ properties, named constants, and API functions. For a complete description, see *FMQCONST copy member* on page 55.

FmqPropvar The FMQPROPVAR copy member provides a complete COBOL definition of the MSMQ propvariant data structure. For an explanation of the propvariant structure, see *Substructures of property structures* on page 68.

The member is recommended for use in programs that create property structures dynamically. For an example of its use, see the *Sample program* on page 74.

FmqLocate The FMQLOCATE copy member defines the data structures used in queue queries.

The member is recommended for use in programs that create the query structures dynamically. For an example, see the FMQBDYNLOC sample program.

Sample programs

The following sample programs, which are located in the SAMPLES file of the Envoy MQ Client library, contain code that illustrates various messaging operations. In particular, the samples illustrate the correct syntax for each API call. You can cut and paste code from the samples, with appropriate modifications, into your programs.

FmqbDyn FMQBDYN is a sample program illustrating the dynamic creation of property structures. The program uses the FMQPROPVAR copy member to define the propvariant data structure.

The program illustrates most of the common messaging operations, such as:

- ❑ Creating and opening a queue
- ❑ Sending and receiving an authenticated message
- ❑ Sending and receiving transacted messages
- ❑ Disconnecting from Envoy MQ Connector

The complete source code of this program is printed in the *Sample program* section of this chapter, page 74.

FmqbStc FMQBSTC is a sample program illustrating basic messaging operations.

The program provides examples of:

- ❑ Connecting to and disconnecting from Envoy MQ Connector
- ❑ Creating and deleting a queue
- ❑ Converting a queue path name to a format name
- ❑ Opening and closing a queue
- ❑ Sending and receiving a message

FmqbDynLoc FMQBDYNLOC is a sample program that creates a queue query dynamically. The program illustrates the use of the FMQLOCATE copy member, and finds a queue having a specified label.

FmqbSample FMQBSAMPLE contains sample API calls for a variety of messaging operations:

- ❑ Creating and closing a cursor
- ❑ Setting and retrieving queue properties
- ❑ Retrieving machine properties
- ❑ Converting a queue handle or GUID to a format name

FMQBSAMPLE is not a complete, compilable program. Rather, it contains fragments of code illustrating the above operations.

Appendix A

RPG Interface for OS/400 V3R2

Chapter 3 of this book, *RPG Interface*, describes the Envoy MQ interface for OS/400 V3R7 or higher.

This appendix describes a functionally identical interface that runs on OS/400 V3R2 or higher. To use this interface, you must install the Envoy MQ Client version for V3R2 (see the *Installation procedure* on page 2).

Differences between the V3R2 and V3R7 interfaces

The only significant difference between the two interfaces is that RPG for V3R2 supports identifiers of up to 10 characters, whereas RPG for V3R7 supports longer identifiers. The RPG interface for V3R7 uses the longer identifiers, which are more similar to the C-language identifiers in the native Envoy MQ and MSMQ APIs.

Programs compiled using the V3R2 interface run on OS/400 V3R2 or higher, including V3R7. Thus if you are programming for a mixed environment of V3R2 and V3R7, you should use the V3R2 interface.

Programs compiled using the V3R7 interface run only on V3R7 or higher. If you are programming for a V3R7 environment, you can use either the V3R2 or V3R7 interface. The V3R7 interface is recommended because the longer identifiers are easier to use.

Other programming topics

The end of this appendix explains the technique for creating null-terminated strings in RPG V3R2 and describes the V3R2 online sample programs and copy members supplied with Envoy MQ Client.

Tables of API identifiers

The Envoy MQ interface for RPG V3R2 is identical to the interface for RPG V3R7 except for the API identifiers. You can translate Envoy MQ code between the two versions by substituting the identifiers.

The following tables list the identifiers for the following API entities:

- ❑ Message properties
- ❑ Queue properties
- ❑ Queue manager properties
- ❑ Value type constants
- ❑ Miscellaneous named constants
- ❑ API functions

✓ *The identifiers are defined in the `FMQCONST` copy members of the two Envoy MQ Client versions. Please refer to the `FMQCONST` source code for other identifiers not listed in the tables.*

Message properties

C	RPG V3R2	RPG V3R7
PROPID_M_ACKNOWLEDGE	P_M_ACK	PID_M_ACK
PROPID_M_ADMIN_QUEUE	P_M_ADM_Q	PID_M_ADMIN_Q
PROPID_M_ADMIN_QUEUE_LEN	P_M_ADM_QL	PID_M_ADMQ_LEN
PROPID_M_APPSPECIFIC	P_M_APPSPC	PID_M_APPSPC
PROPID_M_ARRIVEDTIME	P_M_ARTIME	PID_M_ARVTIME
PROPID_M_AUTH_LEVEL	P_M_AUTHL	PID_M_AUTHTCAT
PROPID_M_AUTHENTICATED	P_M_AUTH	PID_M_AUTH_LVL
PROPID_M_BODY	P_M_BODY	PID_M_BODY
PROPID_M_BODY_SIZE	P_M_BODYL	PID_M_BODY_TYP
PROPID_M_BODY_TYPE	P_M_BODY_T	PID_M_BODY_LEN

C	RPG V3R2	RPG V3R7
PROPID_M_CLASS	P_M_CLASS	PID_M_CLASS
PROPID_M_CONNECTOR_TYPE	P_M_CONTYP	PID_M_CONN_TYP
PROPID_M_CORRELATIONID	P_M_CORRID	PID_M_CORRID
PROPID_M_DELIVERY	P_M_DLVR	PID_M_DELIVERY
PROPID_M_DEST_QUEUE	P_M_DEST_Q	PID_M_DEST_Q
PROPID_M_DEST_QUEUE_LEN	P_M_DEST_L	PID_M_DEST_LEN
PROPID_M_DEST_SYMM_KEY	P_M_SKEY	PID_M_SKEY
PROPID_M_DEST_SYMM_KEY_LEN	P_M_SKEY_L	PID_M_SKEY_LEN
PROPID_M_ENCRYPTION_ALG	P_M_E_ALG	PID_M_ENCR_ALG
PROPID_M_EXTENSION	P_M_EXT	PID_M_EXT
PROPID_M_EXTENSION_LEN	P_M_EXT_L	PID_M_EXT_LEN
PROPID_M_HASH_ALG	P_M_H_ALG	PID_M_HASH_ALG
PROPID_M_JOURNAL	P_M_JRN	PID_M_JOURNAL
PROPID_M_LABEL	P_M_LABEL	PID_M_LABEL
PROPID_M_LABEL_LEN	P_M_LABELL	PID_M_LBL_LEN
PROPID_M_MSGID	P_M_MSGID	PID_M_MSGID
PROPID_M_PRIORITY	P_M_PRTY	PID_M_PRIORITY
PROPID_M_PRIV_LEVEL	P_M_PRIV	PID_M_PRIV_LVL
PROPID_M_PROV_NAME	P_M_PROV	PID_M_PROVN
PROPID_M_PROV_NAME_LEN	P_M_PROV_L	PID_M_PROVN_LN
PROPID_M_PROV_TYPE	P_M_PROV_T	PID_M_PROV_TYP
PROPID_M_RESP_QUEUE	P_M_RES_Q	PID_M_RES_Q
PROPID_M_RESP_QUEUE_LEN	P_M_RES_QL	PID_M_RESQ_LEN
PROPID_M_SECURITY_CONTEXT	P_M_SECNTX	PID_M_SEC_CNTX
PROPID_M_SENDER_CERT	P_M_CERT	PID_M_SNDR_CRT
PROPID_M_SENDER_CERT_LEN	P_M_CERT_L	PID_M_CERT_LEN
PROPID_M_SENDERID	P_M_SID	PID_M_SENDERID

C	RPG V3R2	RPG V3R7
PROPID_M_SENDERID_LEN	P_M_SID_L	PID_M_SID_LEN
PROPID_M_SENDERID_TYPE	P_M_SID_T	PID_M_SID_TYPE
PROPID_M_SENTTIME	P_M_SNTIME	PID_M_SENTTIME
PROPID_M_SIGNATURE	P_M_SIGN	PID_M_SIGN
PROPID_M_SIGNATURE_LEN	P_M_SIGN_L	PID_M_SIGN_LEN
PROPID_M_SRC_MACHINE_ID	P_M_SMCHID	PID_M_SMCH_ID
PROPID_M_TIME_TO_BE_RECEIVED	P_M_T2RCV	PID_M_T2RCV
PROPID_M_TIME_TO_REACH_QUEUE	P_M_T2ARV	PID_M_T2ARV
PROPID_M_TRACE	P_M_TRACE	PID_M_TRACE
PROPID_M_VERSION	P_M_VER	PID_M_VERSION
PROPID_M_XACT_STATUS_QUEUE	P_M_XSTS_Q	PID_M_XSTS_Q
PROPID_M_XACT_STATUS_QUEUE_LEN	P_M_XSTS_L	PID_M_XSTS_QLN

Queue properties

C	RPG V3R2	RPG V3R7
PROPID_Q_AUTHENTICATE	P_Q_AUTHNC	PID_Q_AUTHNCTE
PROPID_Q_BASEPRIORITY	P_Q_BASPRI	PID_Q_BASEPRIO
PROPID_Q_CREATE_TIME	P_Q_CRTIME	PID_Q_CRTIME
PROPID_Q_INSTANCE	P_Q_INSTNC	PID_Q_INSTNC
PROPID_Q_JOURNAL	P_Q_JRN	PID_Q_JRN
PROPID_Q_JOURNAL_QUOTA	P_Q_JQUOTA	PID_Q_JRQUOTA
PROPID_Q_LABEL	P_Q_LABEL	PID_Q_LABEL
PROPID_Q_MODIFY_TIME	P_Q_CHGTME	PID_Q_CHGTIME
PROPID_Q_PATHNAME	P_Q_PATH	PID_Q_PATH

C	RPG V3R2	RPG V3R7
PROPID_Q_PRIV_LEVEL	P_Q_PRVLVL	PID_Q_PRIVLVL
PROPID_Q_QUOTA	P_Q_QUOTA	PID_Q_QUOTA
PROPID_Q_TRANSACTION	P_Q_XACT	PID_Q_XACT
PROPID_Q_TYPE	P_Q_TYPE	PID_Q_TYPE

Queue manager properties

C	RPG V3R2	RPG V3R7
PROPID_QM_CONNECTION	P_C_CONECT	PID_QM_CONNECT
PROPID_QM_ENCRYPTION_PK	P_C_ENCRPT	PID_QM_ENCRYPT
PROPID_QM_MACHINE_ID	P_C_MCH_ID	PID_QM_MCH_ID
PROPID_QM_PATHNAME	P_C_PATH	PID_QM_PATH
PROPID_QM_SITE_ID	P_C_SITEID	PID_QM_SITE_ID

Value type constants

C	RPG V3R2	RPG V3R7
VT_CLSID	VT_CLSID	VT_CLSID
VT_I2	VT_I2	VT_I2
VT_I4	VT_I4	VT_I4
VT_LPWSTR	VT_LPWSTR	VT_LPWSTR
VT_NULL	VT_NULL	VT_NULL
VT_UI1	VT_UI1	VT_UI1
VT_UI2	VT_UI2	VT_UI2

C	RPG V3R2	RPG V3R7
VT_UI4	VT_UI4	VT_UI4
VT_VECTOR | VT_LPWSTR	VT_V#LPWST	VT_VECT#LPWSTR
VT_VECTOR | VT_UI1	VT_V#UI1	VT_VECTOR#UI1

Miscellaneous named constants

The constants are too numerous to list here. For a complete listing, please refer to the FMQCONST source code. The following table provides a few examples.

C	RPG V3R2	RPG V3R7
PSD_SPECIALACCESS_ALL	MQ_ACC_ALL	MQ_ACCESS_ALL
MQ_ERROR_ACCESS_DENIED	MQ_ACC_DND	MQ_ER_ACCESS
MQ_ERROR_BUFFER_OVERFLOW	MQ_BUF_OVR	MQ_ER_BUF_OVR
PRLE	MQ_LE	MQ_LE

API functions

C	RPG V3R2	RPG V3R7
FMQAbort()	FMQAbort	FMQAbort
FMQCommit()	FMQCommit	FMQCommit
FMQConnect()	FMQConnect	FMQConnect
FMQDebug()	FMQDebug	FMQDebug
FMQDisconnect()	FMQDiscon	FMQDisconnect
FMQGetLogPath()	FMQGLogPth	FMQGetLogPath

C	RPG V3R2	RPG V3R7
FMQSetLogPath()	FMQSLogPth	FMQSetLogPath
FMQVersion	FMQVersion	FMQVersion
FMQV1Connect()	FMQV1Cnct	FMQV1Connect()
MQBeginTransaction()	MQBgnTrn	MQBeginTransaction
MQCloseCursor()	MQClsCur	MQCloseCursor
MQCloseQueue()	MQClsQueue	MQCloseQueue
MQCreateCursor()	MQCrtCur	MQCreateCursor
MQCreateQueue()	MQCrtQueue	MQCreateQueue
MQDeleteQueue()	MQDelQueue	MQDeleteQueue
MQFreeMemory()	MQFreeMem	MQFreeMemory
MQFreeSecurityContext()	MQFreSecCt	MQFreeSecurityContext
MQGetMachineProperties()	MQGetMchPr	MQGetMachineProperties
MQGetQueueProperties()	MQGetQProp	MQGetQueueProperties
MQGetSecurityContext()	MQGetSecCt	MQGetSecurityContext
MQHandleToFormatName()	MQHndl2Fmt	MQHandleToFormatName
MQInstanceToFormatName()	MQInst2Fmt	MQInstanceToFormatName
MQLocateBegin()	MQLocBegin	MQLocateBegin
MQLocateEnd()	MQLocEnd	MQLocateEnd
MQLocateNext()	MQLocNext	MQLocateNext
MQOpenQueue()	MQOpnQueue	MQOpenQueue
MQPathNameToFormatName()	MQPath2Fmt	MQPathNameToFormatName
MQReceiveMessage()	MQRcvMsg	MQReceiveMessage
MQRegisterCertificate()	MQRegCer	MQRegisterCertificate
MQSetQueueProperties()	MQSetQProp	MQSendMessage
MQSendMessage()	MQSndMsg	MQSetQueueProperties

Null-terminated strings

MSMQ and Envoy MQ Client require that every string value be terminated by a null character. In RPG, strings are predefined in length and are padded with trailing blanks.

In RPG V3R7, you can convert strings between the two formats using the RPG built-in function `%STR`.

In RPG V3R2, the `%STR` function does not exist. To create a null-terminated RPG string in V3R2, insert `X'00'` at the end of the meaningful text, for example:

```
D M_Label          S            124A
C                    Eval      M_Label = 'Test message ' + X'00'
```

To make sure that the null character is added to the end of the meaningful text, use the built-in function `%TRIM`, for example:

```
D M_Label       S           124A
C                 Eval      M_Label = %TRIM('Test message      ') + X'00'
```

Copy members and sample programs

For a list of RPG copy members and sample programs, see *Online samples* on page 48 in Chapter 3, *RPG Interface*. Envoy MQ Client for OS/400 V3R2 contains versions of the copy members and sample programs for RPG V3R2. The file locations in the Envoy MQ Client V3R2 library are listed in the following table.

File	Description
RPGSRCV3.2	Copy members that you can include in your applications
SAMPLEV3.2	Sample programs illustrating various Envoy MQ programming techniques and solutions to programming problems

Copying Envoy MQ definitions

The following code illustrates how to copy Envoy MQ definitions into an RPG V3R2 program:

```
     D/COPY RPGSRCV3.2,FMQCONST
```

FmqBook program

We recommend that you study the FMQBOOK sample program, which illustrates important messaging operations such as creating queues, sending messages, and receiving messages. The V3R7 version of FMQBOOK is presented in the *Sample program* on page 44. The V3R2 version is almost identical to the V3R7 version, except for the short identifiers.

For an explanation of the programming techniques used in FMQBOOK, see *Data structures (static method)* on page 30.

Envoy MQ Client for AS/400

Index