Developing and Deploying an Application that Accesses MQSeries® Message Queues Using VisualAge ® Generator v4.5

Paul Hoffman, Sanjay Chandru, and Stephen Hancock

Introduction

This paper steps through the use of file level interface for accessing MQSeries message queues. The paper shows how programs are written and systems assembled and deployed at a medium level of detail. For the purposes of discussion, MQ programs described are VisualAge Generator 4.5 programs that access MQSeries message queues as files.

The MQ scenario described in this paper contains client programs on various platforms, two server programs on two platforms, and a database program on one platform. Additionally, two queues are set up, one on each server platform

From three to five operating systems are involved running on workstation and mainframe hardware. The complexities of developing client and server programs on Windows® 95, Windows NT®, and OS/390 platforms are handled by VisualAge Generator and MQSeries. A developer writes the client and server programs in the VisualAge Generator RAD and generates client and server programs.

The two queues described are designed to allow a server to receive messages on one queue and put messages on the other. The two server programs, one on Windows NT and the other on OS/390, can pass messages in this manner while each monitoring just one queue for incoming messages. The queue to which the server program communicates could be running on the OS/390 operating system (a remote queue) although it is known by the Windows NT server program (through a local definition).

Developing the MQ program

Define Target Queue and Queue Manager Objects

In our process of developing an MQ scenario, the first step we take is to define messages in VisualAge Generator. We start by specifying the message format. To do this, we define a message queue record. The record data item structure is the message format. We need to specify the message queue name for this particular record, and we have two options to choose from. We can either define a system resource name for the record in the resource association file, or code the program to move the queue name to the EZEDEST special function word for the message queue record. The format for the queue name in either case is queue-manager-name:queue-name. The queue-manager-name: is optional. If we omit the queue manager name, the default queue manager is used. If we do not specify a queue name for the record, the default queue name is the file name defined in the message queue record properties.

PUT Messages onto a Queue

Once the system is in production, the generated server program on Windows NT receives information from the client programs and puts messages onto a queue. Adding a message to the queue involves using the ADD I/O option in a function with a message queue record as the object. The process automatically connects to the queue manager if there is not an active connection, opens the queue for output if it is not already open, and puts the message on the queue. The server program puts messages by use of a generated MQPUT command. This MQPUT command is equivalent to a manually constructed PUT command in a typical MQSeries program. The MQPUT command results from the developer's use of the VisualAge Generator ADD I/O option.

GET Messages from a Queue

Next, we develop our MQ scenario further to implement message retrieval. The generated OS/390 server program gets the messages from the queue. Our MQ program receives messages by getting them from an incoming message queue. Getting a message from the queue involves using the SCAN I/O option in a function with the message queue record as the object. The process automatically connects to the queue manager if there is not an active connection, opens the queue for input if it is not already open, and gets the message from the queue. The server program gets messages by use of a generated MQGET command. This MQGET command is equivalent to a manually constructed GET command in a typical MQSeries program. The MQGET command results from the developer's use of the VisualAge Generator SCAN I/O option. The generated OS/390 server program passes data using SQL commands to a DB2 database also running on the OS/390 operating system. VisualAge Generator is well suited to handle database transactions and SQL communications. Once operations involving the DB2 database are complete, any returning information is written into messages and put onto a queue running on the Windows NT operating system (a remote queue) known by the OS/390 server program (through a local definition). The Windows NT server gets the messages from the queue and sends information to the appropriate client programs, completing the transaction cycle. We need to address an intermediate level topic in our MQ scenario through message queue record definition. First, we need to specify when the program should open an input queue for exclusive use. We use the *Open queue for exclusive use on input* property of a message queue record to cause the queue to be opened for exclusive use on record SCANs. If we do not check this property in the VisualAge Generator Record Editor user interface, the queue is opened for shared input. When a queue is opened for exclusive input, it is not available to other programs requiring access to the queue object.

CLOSE Connections after Processing

Whether we are sending or receiving messages, we need to make sure our program closes the queue when appropriate. Closing the queue involves using the CLOSE I/O option. We need to close the queue when our program is switching between adding and scanning the same queue, or if we code a long running program that does not need to keep the queue open. Our generated server programs automatically close queues when the program ends or disconnects from the queue manager.

Handle PUTs and GETs to Queues on Multiple Queue Managers

In order to enable the movement of messages, we need to address such issues as connecting to the message queue manager, connecting to a different queue manager, or possibly disconnecting from the message queue manager before the program ends. VisualAge Generator handles the connection to the queue manager automatically on the first ADD or SCAN I/O option in our MQ program, using the queue manager name specified in the system resource name associated with the message queue record. If we do not specify a queue manager name, the default queue manager defined for the production system will be used. In the sample application, this equates to the default queue manager defined for the Windows NT or OS/390 server. VisualAge Generator facilitates automatic disconnection from the queue manager when the program ends, closing any open files and committing the current unit of work if it is still open. VisualAge Generator handles this for us in the generated code. If the connection queue manager and the queue manager to which the queue belongs are different, we code our MQ program to connect before issuing the ADD or SCAN I/O option in the program. We can use the MQCONN reusable part to connect to the appropriate queue manager. The ADD or SCAN I/O option will use the current open connection instead of attempting to connect to the queue manager specified in the system resource name. If our MQ program has already established a connection, the program can access the queue object under a different queue manager by allowing the connected queue manager to communicate with the other queue manger. MQSeries needs to be set up so that each queue manager knows and is able to communicate with other queue managers in the system. Alternatively, we can code an MQDISC call and an MQCONN call if we want to handle the connection at a lower level. To handle the connection at a higher level, our MQ program can include an ADD or SCAN I/O option with the new system resource name under which the queue object exists. If we design a long running program and want to disconnect from the queue manager before the program ends, we can use the MQCONN reusable part to do the initial connection and the MQDISC part to disconnect after all queue access is complete.

Handle Errors

We need our MQ program to verify the movement of messages and validate the integrity of messages. We can accomplish the verification and validation by checking the results of I/O options. Since the function error routine is invoked if the result of any ADD, SCAN, or CLOSE is not completely successful, we can check the results of an I/O option. We can test the record file status values (EOF, ERR, HRD) to determine what kind of result occurred. We can also get the completion code and reason code returned on the MQ API call in the EZERT2 and EZERT8 special function words. Soft errors (reason codes MQRC_NO_MSG_AVAILABLE or MQRC_TRUNCATED_MSG_ACCEPTED) are always returned to the program. We code our MQ program to set EZEFEC to 1 so that all other MQ errors are treated as hard (automatic program termination) errors and are returned to the program.

Perform Transaction Control

To validate messages, we use the *Include message in transaction* property of a message queue record, set in the VisualAge Generator Record Editor user interface, to include the messages in the program's unit of work. We code the program to call EZECOMIT or EZEROLLB to commit or roll back the current unit of work. An implicit commit occurs if the program ends before a commit or rollback is requested. Commits and rollbacks affect both input and output messages. If an input message is rolled back, it goes back on the input queue. If an output message is rolled back, it is not written to the output queue. If our sample application involved transaction environments like CICS, IMS, and AS/400, message queue commits and rollbacks would be coordinated with commits and rollbacks for other recoverable resources, like DB2 databases, using a two-phase commit protocol. For other environments, the resources for different managers are committed independently from one another.

The generated server programs can perform error handling and commit or rollback messages as needed in the course of system operations.

Utilize Variable Length Records

We need to code our MQ program to handle variable length messages. We do this by specifying either a record length item or an occurrences item in the message record properties. The record length item contains the length of the message in bytes. The maximum message length is the length of the message record definition. For ADD I/O options, we code our MQ program to set the item to the desired length before the ADD I/O option is executed. For SCAN I/O options, we code the program to set the item value to the length of the message that is read from the queue. If the message is longer than the record length, the message is truncated. The occurrences item represents the number of valid array entries in the last top level data item in the record structure. The value can range from 0 to the occurs value defined for the last item. The occurrences item must be an item in the message record. The record length item does not have to be defined within the message record. For ADD I/O options, we can code the program to set the occurrences item to the desired number of occurrences before the ADD I/O option is executed. The program calculates the length of the message as the length of the record preceding the last item plus the value of the occurrences item times the length of one occurrence of the array.

Access Advanced MQSeries Options

There are two advanced topics left to address in developing the MQ program, using other MQ API options and specifying different MQ options for different MQ records The MQ Series API Programmer's Reference describes many other options for processing messages besides the options covered by ADD, SCAN, and CLOSE I/O options. These other options are controlled through the setting of options parameters and control blocks passed on the MQOPEN, MQGET, and MQPUT APIs. We can also define advanced options as properties of message queue records. The program would then build the control block and parameters based on these properties. We can code our MQ program to set values in these control blocks to control functions beyond those described above by specifying the names of the MQ options records that we want to override in the advanced options record list in the message queue record properties. The records we specify are automatically included in our program. We can code the program to set options in the records before issuing the ADD or SCAN for the associated message queue record. VisualAge Generator provides working storage records in the MQ reusable parts for MQ control block definitions. The working storage records are MQOD - MQ Object Descriptor, MQOO - MQOPEN options parameter, MQMD - MQ Message Descriptor, MQGMO - MQ Get Message Options, and MQPMO - MQ Put Message Options. If we want our MQ program to specify more than one copy of an options record for use with different message queue records in the same program, we can define new records as alternate specifications of the options records. Alternate specification is defined in a property of the new record and specifies the same data item definition as the record named in the property.

Triggering and MQSeries

MQSeries triggering capabilities allow programs to be started when a message is put on a particular queue during a specified length of time. If a server, such as the OS/390 in the example above, uses triggering, system resources can be saved by starting the server program only when the Windows NT server program puts a message on its outbound queue.

Deploying the MQ program

There are run-time considerations for our MQ program that we need to address. An important aspect of this being deployment of the program in specific run-time environments, which may involve

- Converting data formats
- Using resource associations
- Using linkage tables for MQSeries run-time library selection

Deploying the program in specific run-time environments

The first consideration we need to address for deploying our MQ program involves deployment in specific run-time environments. To handle specific run-time environment considerations, we can use a resource association file to specify a default queue name and conversion table for a message queue record in a specific environment. We use a linkage table to specify which MQSeries library (client or server, threaded or unthreaded) we want to use in the specific environment.

Data Format Conversions

To handle data format conversion, we specify a conversion table name in the resource association file entry for the message queue record. We need to do this if we code our MQ program to send and receiving messages between systems that use different code pages for character data or different numeric data formats. We need to specify the conversion table when generating either the client program or the server program, but not both. In our sample application, we specify the appropriate conversion table (ELACNENU for English) in the resource association file when generating the MQ program. The messages added to the outbound queue are translated from Windows NT to OS/390 format and messages from the inbound queue are converted back from OS/390 format to Windows NT format. No conversion is necessary for the server program running on OS/390 since the messages are already in OS/390 format on the queue.

Use of Resource Associations

We use resource associations to specify queue name and data format conversion. Since *File name* is a required property in a message record definition, resource associations are specified by file name and not by record name. If we define message queue records with the same file name, they share the same resource associations. We use a resource association for the VisualAge Generator Test Facility just like we use a resource association for any other specific environment. To add a resource association entry for test facility, we type the file name in the *Logical name* field, select *Message Queue* as the *Organization*, and type the queue manager/queue name in the *Physical Name* field. We format the queue manager name and queue name in the resource association just like we would if we were moving the queue name to EZEDEST, queue_manager_name:queue_name. The queue manager name specified here is the queue manager associated with the queue. If we want data format conversion performed when messages are added to the queue or read from the queue, and we do in our sample application, we need to type the conversion table name in the *Conversion Table* field. When we add a resource association entry for a message queue record to a resource association part for generation and run time, we specify the entry as follows:

```
ASSOCIATE  FILE=filename/FILETYPE=MQ
           /SYSNAME=queue_manager_name:queue_name
           /CONTABLE=conversion_table_name
           /SYSTEM=target_system
```

We can omit CONTABLE if we do not want data format conversion performed for the message. We can also omit SYSTEM if we want the entry to apply to any system. We need to specify the resource association part when running C++ programs generated for non-CICS environments. For all other environments, we specify the resource association part at generation using the /RESOURCE generation option.

Use of Linkage Tables

In workstation environments (NT, OS/2, AIX, Solaris, and HP), MQSeries provides different runtime libraries for MQ programs depending on whether the program is running on the same system as the message queue manager or whether the program is running as an MQ client communicating with a manager on a server system. On AIX and HP systems, different libraries are provided for threaded and non-threaded environments as well. We use a VisualAge Generator linkage table part to indicate which runtime library we want to use. The MQ reusable parts shipped with VisualAge Generator include sample linkage tables for all supported environments. We can use these parts directly, or copy the entries in the parts to our own linkage table, if we need to specify entries for other program calls. If we test or run our MQ program with an MQ manager, non-threaded library, we specify the linkage table part as a test or generation option. If we test or run our MQ program with an MQ client or threaded library, we must also move the part to a file and set the CSOLINKTBL environment variable to the file name. If we generate our MQ program in Java™, it requires a special format for the linkage table entry. The entry should look like the following:

```
:calllink applname=elaq* library=mq_wrapper_dll_name linktype=csocall
          parmform=commptr remotecomtype=direct remoteapptype=nonvg
          contable=java_conversion_table_name
```

where the mq_wrapper_dll is the wrapper dll name for our runtime environment, and the java_conversion_table_name is the Java conversion table correct for our language and the system on which the program is running.

Summary

Developing an MQ application with VisualAge Generator 4.5 is simple and efficient. MQ support utilizes existing I/O options, error handling, and transaction control. This virtually eliminates the need for existing customers to learn complicated new concepts to effectively deploy MQSeries applications on their systems. A concerted attempt has been made to explain an actual deployment scenario in detail. This paper also illustrates the ease of deployment in several different platforms, which effectively exploits the middleware paradigm of MQSeries.

Information and Reference

For general information on MQSeries and VisualAge Generator:
Http://www-4.ibm.com/software/ts/mqseries/
Http://www-4.ibm.com/software/ad/visgen/
Http://www.redbooks.ibm.com/; MQSeries Primer, REDP0021

For specific information on implementing MQSeries programs in VisualAge Generator, refer to Chapter 4, Developing MQSeries Application Systemspp.77-pp.107 in the VisualAge Generator Version 4.5 User's Guide.

Trademarks

Produced 3-2001