7

MQ update

January 2000

In this issue

MQ Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: +44 1635 550955
e-mail: HarryL@xephon.com

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
USA
Telephone: +1 940 455 7050

Contributions

Articles published in *MQ Update* are paid for at the rate of £170 ($250) per 1000 words and £90 ($140) per 100 lines of code. For more information about contributing an article, please check Xephon's Web site, where you can download *Notes for Contributors*.

MQ Update on-line

Code from *MQ Update* is available from Xephon's Web site at www.xephon.com/mqupdate.html (you'll need the user-id shown on your address label to access it). If you've a problem with your user-id or password call Xephon's subscription department on +44 1635 33886.

Editor

Harry Lewis

Subscriptions and back-issues

A year's subscription to *MQ Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the July 1999 issue, are available separately to subscribers for £22.50 ($33.50) each including postage.

The Application Messaging Interface

INTRODUCTION

Application developers have become accustomed to using the MQSeries Interface (MQI) for enterprise application integration. MQI is a simple interface with which to begin developing messaging applications – for instance, there are only thirteen MQI calls to learn. All this apparent simplicity comes at a cost, the cost being the incredible number of options available under MQI. This makes MQI a rich, low-level messaging API. However, application developers often protest at being overwhelmed with MQI's details, which they consider unrelated to business logic. This is also a problem for MQSeries administrators as MQI allows access to almost all aspects of the middleware. Another common problem is that, once the application is deployed, changing any of its attributes, such as message persistence, priority, and wait time for receiving applications, needs redevelopment and redeployment.

Application architects and developers address these issues by designing wrapper APIs that hide the complexity of the MQI from application developers. These wrappers usually aren't general enough to accommodate centralized control of the messaging details. They are also organization-specific and fail adequately to distinguish the duties of MQSeries administrators from those of application developers in a logical way. Application developers also face the challenge of learning several messaging interfaces, if they need to develop applications that use several messaging products.

Therefore, a need exists for a messaging API that provides both a higher level of message abstraction and standardization. The Open Applications Group (*www.openapplications.org*), which is a non-profit consortium of leading enterprise software developers, realized this and worked on the requirements for a standard messaging API. Recently, it accepted IBM's proposal for the Application Messaging Interface to be adopted as the Open Application Middleware API specification. IBM has quickly followed up the AMI specification with an implementation via support packs *MA0F* and *MA0G*.

In this article, I introduce the AMI and its underlying model. I then explore the API and its use for developing standard messaging applications. The article contains working samples of application programs that demonstrate significant differences between the MQI and AMI.

THE MAIN FEATURES OF THE AMI

The AMI has three main components:

- The message
- The service
- The policy.

The message is what is being sent (or received). The service is defined as the location where the message is sent (or received). The policy determines how the message is sent or received.

For the purpose of mapping these components to the MQI, an AMI message is a 'most probably' subset of an MQI message. In other words, there are several attributes that are available in the MQI that are not available in a message as defined by the AMI. One can't really say that an AMI message is a subset of an MQI message as the implementation of an AMI message is not visible.

A service can be mapped onto a queue and queue manager combination. There is more to a service than just a queue name, though a queue name is a good approximation of a service.

A policy is a new concept that takes some attributes from MQI messages and some from MQI applications and makes them available at a central location. This separation of the policy from the message is one of the greatest strengths of the AMI, making it a very attractive interface, even if MQSeries is currently the only messaging middleware in use in the organization.

In order to use the AMI, developers should specify the message data, the service, and the policy. Developers can choose to use the default services and policy that are provided with the AMI. MQSeries administrators can also create additional services and policies, which may then be stored in a repository. Support pack *MA0F* provides a default implementation of the repository.

AMI currently supports basic messaging applications that:

- Send a message
- Receive a message
- Request/respond
- Publish/subscribe.

The API provision of request/response facilities satisfies a pent-up demand for this functionality. Application developers have always been able to develop this type of application before by managing the message id/correlation id. However, this is not an intuitive way of doing things. On the other hand, the AMI doesn't seem to have support for other parts of the MQSeries framework, including the Trigger Monitor Interface and the Message Channel Interface (Publish-and-Subscribe is a new area of functionality in MQSeries 5.1). Nevertheless, the AMI is interoperable with other MQSeries interfaces – you can exchange messages with other applications that use MQI using the AMI.

The AMI is currently available in C, C++, and Java. For the C version, the AMI is available in two 'levels': a high-level procedural interface and a low-level object interface.

AMI COMPONENTS

The AMI requires three main components: the message, the service, and the policy. In this section I provide a more detailed definition of each component.

The message

The message comprises message attributes and message data. As far as the AMI is concerned, message data is application data and the AMI does not act on it. Message attributes, such as the *MessageID* and *CorrelID*, are properties of the message itself. The AMI uses properties of the message object, along with the policy, to construct MQSeries headers and descriptors. Message attributes can be mapped to MQMD members. However, not all MQMD fields are available in the message object – some, such as persistence and expiry, are not attributes of the AMI message. The AMI lets you name message objects, which is a useful feature. Developers have to use message names in request/response-type applications to match request and response messages.

The AMI then has functions to set and get each of the message attributes and functions to read and write message data.

The service

To the AMI, a service is a location where a message can be sent or received. In MQSeries terminology, a destination is a queue residing in a queue manager. The service is set up by an MQSeries administrator and, consequently, the complexity of setting up and managing queues is hidden from application programmers. There are five types of service available via the AMI:

1 Sender

2 Receiver

3 Distribution list

4 Publisher

5 Subscriber.

A *sender* service establishes one-way communication for sending messages. A *receiver* service establishes one-way communication for receiving messages. A *distribution list* contains a list of senders to which messages can be sent. A *publisher* contains a sender that publishes messages to the publish/subscribe broker. A *subscriber* contains a sender, for subscribing to a broker, and a receiver, to get publications from a broker.

The AMI has functions explicitly to open and close services. Services can also be opened and closed implicitly by other functions. Functions are also provided for exception processing. Services use policy objects to get the correct MQSeries options.

The AMI provides a default example of each type of service. Application programmers can use the AMI's default services or ones provided by the MQSeries administrator. Administrators use the AMI Admin tool to set up and manage custom MQSeries services, and the custom services they set up can be stored in a repository.

The current implementation of the repository does not appear to distinguish a sending service from a receiving service. As a consequence, one can use a sending service in a receive-only type application. This means that the distinction between a sending and receiving service depends on the context in this implementation.

The policy

The policy is one of the most interesting aspects of the AMI, and it controls how AMI functions operate. Policies control items relating to the message object and service objects. For example, the priority and persistence of the message are controlled by the policy. In the case of sender and receiver services, the policy defines whether together they participate in a unit of work. The policy also determines the retry options for services.

It’s possible for each AMI call to use a different policy. For example, when an application exchanges the same message with a number of other applications, each message can be sent using a policy that’s linked to the recipient. The other approach would be to use one policy that’s shared by every call in an application. This would make a policy change impact the behaviour of the entire application. This kind of flexibility can be provided with policies.

In common with services, some default policies are provided with the AMI. Policies can be customized and stored in a repository, with the AMI Admin Tool being used to create and manage them.

Other AMI objects

There are two other AMI objects of which one should be aware: the *session* and *connection* objects. The session object is a container for all message, service, and policy objects. It contains a connection object that’s not visible to applications. The session object creates and manages other objects, and also provides the default scope of a unit of work. AMI contains functions to create, initialize, open, close, and terminate a session. Within a session, AMI provides an API to create, manage, and destroy other objects. Functions are provided to control transaction processing and error handling.

THE AMI API

As mentioned earlier, the AMI supports two APIs for C. The high-level C API comprises the thirteen functions listed below (in this article, we concentrate on the first seven of them).

- Session management:
  - *amInitialize*
  - *amTerminate*.
- Send message:
  - *amSendMsg*
  - *amSendRequest*
  - *amSendResponse*.
- Receive message:
  - *amReceiveMsg*
  - *amReceiveRequest*.
- Publish/subscribe
  - *amPublish*
  - *amSubscribe*
  - *amUnsubscribe*
  - *amReceivePublication*.
- Transaction support:
  - *amCommit*
  - *amBackout*.

The low-level APIs provide greater access to constructors, destructors, ‘set’ and ‘get’ functions for individual messages, and service and policy objects.

AMI IMPLEMENTATION

The AMI implementation comprises language support for AMI-specified functions and repository support for storing services and policies. Services and policies may be stored in XML files. Support pack *MA0G* provides a GUI interface to an XML repository, though (at the time of writing) only an NT version was released. This AMI Administration Tool is used to create new policies and services and maintain the repository.

ENVIRONMENT

In this article, we look at the details of the AMI and its current implementation. The samples provided with this article use MQSeries 5.1 on Windows NT 4 (a minimum of SP3 is recommended). As mentioned earlier, support packs *MA0F* and *MA0G* should be installed.

APPLICATION DEVELOPMENT IN AMI

To illustrate application development in C using the high-level API, we develop application programs that demonstrate AMI concepts and make use of the salient features of the API. Hence, we develop small programs to:

- Send a message to sender service *SAMPLE.SENDER* (see *samplesend.c*).
- Receive a message from receiver service *SAMPLE.RECEIVER* (see *samplercv.c*).
- Carry out request/response type processing (*sampleclt.c* illustrates client-side processing and *samplesvr.c* server-side processing).

Administrative actions

To achieve these objectives, we need to carry out the following administrative activities:

- Set up a sender service called *SAMPLE.SENDER*. For this, you need to specify at least the queue name. Let's call the queue *SAMPLE.REQUEST.QUEUE*, as this is the name used in *samplesend.c*.

- Set up a receiver service called *SAMPLE.RECEIVER.* The queue name specified is *SAMPLE.REQUEST.QUEUE* (the name used in *samplercv.c*).
- Set up a sender service called *REQUEST.SERVICE* with queue name *SAMPLE.REQUEST.QUEUE* (the name used in *sampleclt.c*).
- Set up a receiver service *RESPONSE.SERVICE* with queue name *SAMPLE.RESPONSE.QUEUE* (the name used in *sampleclt.c*).
- Set up a receiver service *REQUEST.RECEIVE.SERVICE* with queue name *SAMPLE.REQUEST.QUEUE* (the name used in *samplesvr.c*).
- Set up a policy called *SAMPLE.POLICY.*
- Create queue *SAMPLE.REQUEST.QUEUE* in the default queue manager.
- Create queue *SAMPLE.RESPONSE.QUEUE* in the default queue manager.

As mentioned before, several MQMD-related message attributes and MQPUT options are controlled by means of the sender service and policy.

Sending a message using the AMI

The application has following structure:

- Create and initialize session
- Send message
- Terminate the session.

Unlike an MQI application, there are no explicit *MQCONN* and *MQDISC* calls and their associated drawbacks (for example, *MQCONN* needs to be passed the name of the queue manager, which is usually coded in the application). The AMI model stores the queue manager name in the sender service, which means that no application changes are necessary if the queue manager name in *MQCONN* needs to be changed.

The calls used in the sample program are:

```
hSession = amInitialize (name, myPolicy, pCompCode, pReason)
```

The above call creates and opens the named session.

```
success = amSendMsg (hSession, mySender, myPolicy, dateLen,
➤  pData,NULL, pCompCode, pReason)
```

The above call creates a sender object, a policy object, and a message object, and then invokes 'send' (note the use of the continuation character, '➤', to indicate that one line of code maps to more than one line of print).

```
success = amTerminate (&hSession, myPolicy,  pCompCode, pReason)
```

This call closes and deletes the session. The process of deleting a session, in turn, implicitly closes and deletes the message object, sender object, and the policy object.

amInitialize returns a handle to session object. *amSendMessge* and *amTerminate* both return an *AMBOOL* type that may take the value *AMB_TRUE* or *AMB_FALSE*.

The *include* file *amtc.h* defines all function prototypes and constants, including *AMB_TRUE*, *AMB_FALSE*, and *AMH_NULL_HANDLE*.

Receiving a message using AMI

The receiver application is very similar at the conceptual level. Obviously you need a receiver service, *SAMPLE.RECEIVER*, to pick up messages from *SAMPLE.REQUEST.QUEUE*.

We continue to use both *amInitialize* and *amTerminate* calls, and additionally use *amReceiverMsg* calls:

```
success = amReceiveMsg (hSession, myReceiver, myPolicy,
➤  selMessageName, dataLen, pData, rcvMsgName, pCompCode,
➤  pReason)
```

Here, *selMessageName* is a message object that can be constructed using the low-level AMI API for C. It is typically used for selecting messages based on correlation id. In our example, we use *NULL* as our selection criterion (in other words, we use the default selection criterion).

Request/response-type processing using AMI

The AMI provides special support for request/response-type applications. Our client program is structured as follows:

- Create and open session (*amInitialize*)
- Send request (*amSendRequest*)
- Receive response, possibly with wait (*amReceiveMsg*)
- Terminate session (*amTerminate*).

amSendRequest differs from *amSendMsg* in that it allows you to specify a receiver to whom the response is sent. The sender message name is then used with an *amReceiveMsg* call to select the matching response.

```
success= amSendRequest (hSession, mySender, myPolicy,
➤  myReceiver, dateLen, pData, NULL, pCompCode, pReason)
```

In the *amSendRequest* call, *mySender* and *myReceiver* are existing service points (the *NULL* default specification is accepted). If either doesn’t exist, you get an error when the code completes.

The server program is structured as follows:

- Create and open a session (*amInitialize*)
- Receive a request (*amReceiveRequest*)
- Send response (*amSendResponse*)
- Terminate session (*amTerminate*).

amReceiveRequest takes the name of the sender service that is used for the response. Unlike the *amReceiveMsg* call, it doesn’t support selection criteria. The response is sent using an *amSendResponse* call. This causes the *CorrelId* and *MessageId* to be set in the response message, based on the flags in the request message. The *amSendResponse* call uses the sender service and the response message name specified by the request message received by *amReceiveRequest*.

```
Success = amReceiveRequest (hSession, myReceiver, myPolicy,
➤  buffLen, pDataLen, pData, rcvMsgName, mySender, pCompCode,
➤  pReason)
```

In an *amReceiveRequest* call, the name of *mySender* should be specified as one that doesn't exist in the AMI repository. This allows the request message to dictate which sender service should be used for the response. If *mySender* exists in the repository, you will receive a completion code of '2' and a reason code of '70'.

```
success = amSendResponse (hSession, mySender, myPolicy,
➤  rcvMsgName, dataLen, pData, sndMsgName, pCompCode, pReason)
```

The sender name for the *amSendResponse* call is the same as that specified in the *amReceiveRequest* call.

SAMPLESEND.C

```
/*****************************************************************/
/* File Name: samplesend.c                                       */
/* Purpose: Send a message to the sender service SAMPLE using    */
/*          policy SAMPLE.POLICY.                                */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <amtc.h>

#define SAMPLE_SESSION_NAME  "SAMPLE.SESSION"
#define SAMPLE_SENDER_NAME   "SAMPLE.SENDER"
#define SAMPLE_MESSAGE_NAME  "SAMPLE.SEND.MESSAGE"
#define SAMPLE_POLICY_NAME   "SAMPLE.POLICY"
#define SAMPLE_MESSAGE_DATA  "Application data for sample send program"

int main(int argc, char** argv)
{
  AMHSES   hSession   = AMH_NULL_HANDLE;
  AMLONG   compCode;
  AMLONG   reason;
  AMBOOL   success    = AMB_FALSE;
  AMHMSG   hMessage   = AMH_NULL_HANDLE;
  char     sampleMsg[256];

/*****************************************************************/
/* Initialize (create and open) the session with the specified   */
/* name.                                                         */
/*****************************************************************/

printf("Starting samplesend program \n");
hSession = amInitialize( SAMPLE_SESSION_NAME     /* session name    */
                        ,SAMPLE_POLICY_NAME      /* policy          */
                        , &compCode              /* completion code */
```

```
                         , &reason);             /* reason code     */
  if ( hSession == AMH_NULL_HANDLE )
  {
    printf("*** amInitialize() failed cc = %d, rc = %d \n",
            compCode, reason);
    printf(" Completed samplesend \n");
    return EXIT_FAILURE;
  }
  printf("amInitialize() succeeded \n");

/*******************************************************************/
/* Send the message using the sample data.                         */
/*******************************************************************/
  sprintf(sampleMsg, "%s", SAMPLE_MSG_DATA);

/*******************************************************************/
/* Send the message as a datagram using amSendMsg.                 */
/*******************************************************************/
  success = amSendMsg( hSession                  /* session handle  */
           , SAMPLE_SENDER_NAME                  /* sender          */
           , SAMPLE_POLICY_NAME                  /* policy          */
           , strlen(sampleMsg)+1                 /* data length     */
           , (unsigned char *)sampleMsg          /* message buffer  */
           , SAMPLE_MESSAGE_NAME                 /* message object  */
           , &compCode                           /* completion code */
           , &reason );                          /* reason code     */
  if ( success == AMB_FALSE )
  {
    printf("*** amSendMsg() failed cc = %d, rc = %d \n",
            compCode, reason );
    amTerminate( &hSession, SAMPLE_POLICY_NAME, &compCode, &reason);
    printf(" Completed samplesend \n");
    return EXIT_FAILURE;
  }
  printf("    amSendMsg() succeeded \n");

/*******************************************************************/
/* Terminate the session.                                          */
/*******************************************************************/
  success = amTerminate( &hSession               /* session handle  */
                       , "SAMPLE.POLICY"         /* policy          */
                       , &compCode               /* completion code */
                       , &reason);               /* reason code     */
  if ( success == AMB_FALSE )
  {
    printf("*** amTerminate() failed cc = %d, rc = %d \n",
           compCode, reason);
    printf(" Completed samplesend \n");
    return EXIT_FAILURE;
  }
  printf("    amTerminate() succeeded \n");

  printf(" Completed samplesend\n");
  return EXIT_SUCCESS;
}
```

SAMPLERCV.C

```
/*********************************************************************/
/* File name: samplercv.c                                            */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <amtc.h>

#define SAMPLE_SESSION_NAME      "SAMPLE.SESSION"
#define SAMPLE_RECEIVER_NAME     "SAMPLE.RECEIVER"
#define SAMPLE_POLICY_NAME       "SAMPLE.POLICY"
#define SAMPLE_MESSAGE_NAME      "SAMPLE.RECEIVE.MESSAGE"

int main(int argc, char **argv)
{
  AMLONG   compCode;
  AMLONG   reason;
  AMHSES   hSession    = AMH_NULL_HANDLE;
  AMBOOL   success     = AMB_FALSE;
  AMLONG   dataLength;
  AMLONG   msgCount    = 0;
  char     data[256];

/********************************************************************/
/* Initialize (create and open) the session with the specified      */
/* name.                                                            */
/********************************************************************/
  printf("<<< Starting samplercv >>>\n");
  hSession = amInitialize( SAMPLE_SESSION_NAME   /* session name    */
                         , "SAMPLE.POLICY"       /* policy          */
                         , &compCode             /* completion code */
                         , &reason);             /* reason code     */
  if ( hSession == AMH_NULL_HANDLE )
  {
    printf("*** amInitialize() failed cc = %d, rc = %d \n",
           compCode, reason);
    printf("<<< Completed samplercv >>>\n");
    return EXIT_FAILURE;
  }
  printf("    amInitialize() succeeded \n");
/*********************************************************************/
/* Receive the message using amReceiveMsg.                           */
/*********************************************************************/
    success = amReceiveMsg( hSession       /* session handle        */
                   , SAMPLE_RECEIVER_NAME  /* receiver service name */
                   , "SAMPLE.POLICY"       /* policy                */
                   , NULL                  /* No selection message  */
                   , sizeof(data)          /* buffer length         */
                   , &dataLength           /* data length           */
                   , (unsigned char *)data /* messsage data         */
                   , SAMPLE_MESSAGE_NAME   /* message               */
                   , &compCode             /* completion code       */
                   , &reason );            /* reason code           */
    if ( success == AMB_FALSE )
    {
      printf("*** amReceiveMsg() failed cc = %d, rc = %d \n",
              compCode, reason);
      amTerminate( &hSession, SAMPLE_POLICY_NAME, &compCode, &reason);
      printf("<<< Completed samplercv >>>\n");
      return EXIT_FAILURE;
    }
    data[dataLength] = '\0';
    printf("      %s\n", data);

/*********************************************************************/
/* Terminate the session.                                            */
/*********************************************************************/
  success = amTerminate( &hSession               /* session handle  */
                       , "SAMPLE.POLICY"         /* policy          */
                       , &compCode               /* completion code */
                       , &reason);               /* reason code     */
  if ( success == AMB_FALSE )
  {
    printf("*** amTerminate() failed cc = %d, rc = %d \n",
            compCode, reason);
    printf("<<< Completed samplercv >>>\n");
    return EXIT_FAILURE;
  }
  printf("    amTerminate() succeeded \n");

  printf("<<< Completed samplercv >>>\n");
  return EXIT_SUCCESS;
}
```

SAMPLECLT.C

```
/**********************************************************************/
/* File name: sampleclt                                               */
/**********************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <amtc.h>

#define SAMPLE_SESSION_NAME         "REQUEST.SESSION"
#define SAMPLE_POLICY_NAME          "SAMPLE.POLICY"
#define SAMPLE_SENDER_NAME          "REQUEST.SERVICE"
#define SAMPLE_RECEIVER_NAME        "RESPONSE.SERVICE"
#define SAMPLE_SEND_MESSAGE_NAME    "REQUEST.MESSAGE"
#define SAMPLE_RECEIVE_MESSAGE_NAME "RESPONSE.MESSAGE"

#define SAMPLE_MSG_DATA             " REQUEST "

int main( int argc , char ** argv )
{
  AMLONG    compCode;
  AMLONG    reason;
  AMHSES    hSession;
  AMBOOL    success;
  char      request[1024];
  char      reply[1024];
  AMLONG    dataRead = 0;

/********************************************************************/
/* Initialize (create and open) the session with the specified      */
/* name.                                                            */
/********************************************************************/
  printf(" Starting sampleclt \n");
  hSession = amInitialize( SAMPLE_SESSION_NAME   /* session name    */
                         , SAMPLE_POLICY_NAME    /* policy          */
                         , &compCode             /* completion code */
                         , &reason);             /* reason code     */
  if ( hSession == AMH_NULL_HANDLE )
  {
    printf("*** amInitialize() failed cc = %d, rc = %d \n",
            compCode, reason);
    printf(" Completed sampleclt \n");
    return EXIT_FAILURE;
  }
  printf("    amInitialize() succeeded \n");

  memset(request, 0, sizeof(request));
  sprintf(request, "sampleclt%s\0", SAMPLE_MSG_DATA );

/********************************************************************/
/* Send the request message using amSendRequest.                    */
/********************************************************************/
  success = amSendRequest( hSession          /* session handle      */
                 , SAMPLE_SENDER_NAME        /* sender service name */
```

```
                , SAMPLE_POLICY_NAME        /* policy               */
                , SAMPLE_RECEIVER_NAME      /* receiver svc name    */
                , strlen(request)+1         /* data length          */
                , (unsigned char *)request  /* messsage data        */
                , SAMPLE_SEND_MESSAGE_NAME  /* message name         */
                , &compCode                 /* completion code      */
                , &reason );                /* reason code          */
  if ( success == AMB_FALSE )
  {
    printf("*** amSendRequest() failed cc = %d, rc = %d \n",
           compCode, reason);
    amTerminate( &hSession, SAMPLE_POLICY_NAME, &compCode, &reason);
    printf(" Completed sampleclt \n");
    return EXIT_FAILURE;
  }
  printf("      SENT\n");
  printf("       %s\n", request);

/*******************************************************************/
/* Receive the response message using amReceiveMsg.               */
/*******************************************************************/
  success = amReceiveMsg( hSession          /* session handle       */
             , SAMPLE_RECEIVER_NAME         /* receiver svc name    */
             , SAMPLE_POLICY_NAME           /* policy name          */
             , SAMPLE_SEND_MESSAGE_NAME     /* sel message name     */
             , sizeof(reply)                /* buffer length        */
             , &dataRead                    /* returned data length */
             , (unsigned char *)reply       /* messsage data        */
             , SAMPLE_RECEIVE_MESSAGE_NAME  /* reply message name   */
             , &compCode                    /* completion code      */
             , &reason );                   /* reason code          */
  if ( success == AMB_FALSE )
  {
    if ( reason == AMRC_NO_MSG_AVAILABLE )
    {
      printf("*** REPLY did not arrive within the specified policy \
            wait time\n" );
    }
    else
    {
      printf("amReceiveMsg() failed cc = %d, rc = %d \n",
            compCode, reason);
      amTerminate( &hSession, SAMPLE_POLICY_NAME, &compCode, &reason);
      printf(" Completed sampleclt \n");
      return EXIT_FAILURE;
    }
  }
  reply[dataRead] = '\0';
  printf("      RECEIVED\n");
  printf("       %s\n", reply);
```

```
/*********************************************************************/
/* Terminate the session.                                            */
/*********************************************************************/
  success = amTerminate( &hSession               /* session handle  */
                       , SAMPLE_POLICY_NAME      /* policy          */
                       , &compCode               /* completion code */
                       , &reason);               /* reason code     */
  if ( success == AMB_FALSE )
  {
    printf("*** amTerminate() failed cc = %d, rc = %d \n",
            compCode, reason);
    printf(" Completed sampleclt \n");
    return EXIT_FAILURE;
  }
  printf("    amTerminate() succeeded \n");

  printf(" Completed sampleclt \n");
  return EXIT_SUCCESS;
}
```

SAMPLESVR.C

```
/*********************************************************************/
/* File name: samplesvr                                              */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <amtc.h>
#define SAMPLE_SESSION_NAME         "RESPONSE.SESSION"
#define SAMPLE_POLICY_NAME          "SAMPLE.POLICY"
#define SAMPLE_RECEIVER_NAME        "REQUEST.RECEIVE.SERVICE"
#define SAMPLE_SENDER_NAME          "RESPONDER.SERVICE"
#define SAMPLE_SEND_MESSAGE_NAME    "RESPONSE.MESSAGE"
#define SAMPLE_RECEIVE_MESSAGE_NAME "REQUEST.MESSAGE"

int main(int argc, char **argv)
{
  AMLONG   compCode;
  AMLONG   reason;
  AMHSES   hSession   = AMH_NULL_HANDLE;
  AMBOOL   success    = AMB_FALSE;
  AMLONG   dataRead   = 0;
  AMLONG   msgCount   = 0;
  char     request[1024];
  char     reply[1024];

/*********************************************************************/
/* Initialize (create and open) the session with the specified      */
/* name.                                                             */
/*********************************************************************/
  printf(" Starting samplesvr \n");
  hSession = amInitialize( SAMPLE_SESSION_NAME   /* session name    */
                         , SAMPLE_POLICY_NAME    /* policy          */
                         , &compCode             /* completion code */
                         , &reason);             /* reason code     */
  if ( hSession == AMH_NULL_HANDLE )
  {
    printf("*** amInitialize() failed cc = %d, rc = %d \n",
            compCode, reason);
    printf(" Completed samplesvr \n");
    return EXIT_FAILURE;
  }
  printf("    amInitialize() succeeded \n");

/*********************************************************************/
/* Receive requests.                                                 */
/*********************************************************************/
    success = amReceiveRequest( hSession         /* session handle  */
                  , SAMPLE_RECEIVER_NAME         /* receiver name   */
                  , SAMPLE_POLICY_NAME           /* policy          */
                  , sizeof(request)              /* buffer length   */
                  , &dataRead                    /* data length     */
                  , (unsigned char *)request     /* messsage data   */
                  , SAMPLE_RECEIVE_MESSAGE_NAME  /* message name    */
                  , SAMPLE_SENDER_NAME           /* response sender */
                  , &compCode                    /* completion code */
                  , &reason );                   /* reason code     */
    if ( success == AMB_FALSE )
    {
      printf("*** amReceiveRequest() failed cc = %d, rc = %d \n",
              compCode, reason);
      amTerminate( &hSession, SAMPLE_POLICY_NAME, &compCode, &reason);
      printf(" Completed samplesvr \n");
      return EXIT_FAILURE;
    }
    else
    {
      printf("       RECEIVED request\n");
      request[dataRead]='\0';
      printf("        %s\n", request);
    }

/**********************************************************************/
/* Send the reply.                                                    */
/**********************************************************************/
    sprintf(reply, "samplesvr REPLY " );
    success = amSendResponse( hSession         /* session handle     */
                 , SAMPLE_SENDER_NAME          /* sender svc name    */
                 , SAMPLE_POLICY_NAME          /* policy              */
                 , SAMPLE_RECEIVE_MESSAGE_NAME /* request msgname     */
                 , 1+strlen(reply)             /* data length         */
                 , (unsigned char *)reply      /* messsage data       */
                 , SAMPLE_SEND_MESSAGE_NAME    /* reply message name */
                 , &compCode                   /* completion code    */
                 , &reason );                  /* reason code        */
    if ( success == AMB_FALSE )
    {
      printf("*** amSendResponse() failed cc = %d, rc = %d \n",
              compCode, reason);
      amTerminate( &hSession, SAMPLE_POLICY_NAME, &compCode, &reason);
      printf(" Completed samplesvr \n");
      return EXIT_FAILURE;
    }
    printf("       SENT reply\n");
    printf("        %s\n", reply);

/**********************************************************************/
/* Terminate the session.                                             */
/**********************************************************************/
  printf("Calling amTerminate \n");
  success = amTerminate( &hSession                /* session handle  */
                       , SAMPLE_POLICY_NAME       /* policy          */
                       , &compCode                /* completion code */
                       , &reason);                /* reason code     */
  if ( success == AMB_FALSE )
  {
    printf("amTerminate() failed cc = %d, rc = %d \n",
            compCode, reason);
    printf(" Completed samplesvr \n");
    return EXIT_FAILURE;
  }
  printf("    amTerminate() succeeded \n");

  printf(" Completed samplesvr \n");
  return EXIT_SUCCESS;
}
```

COMPILE AND LINK

Using Visual C 98 running on Windows NT, the commands needed to compile and link are:

```
set lib=%lib%;d:\mq\bin
d:\program files\microsoft visual studio\vc98\bin\vcvars32.bat
cl -ID:\mq\amt\include /Fsamplesend samplesend.c  amt.LIB
```

Other programs are compiled by replacing the C files and executables.

SUMMARY

In this article, I introduced the Application Messaging Interface, which is an emerging alternative to the MQI. The AMI has several advantages over MQI: it provides a higher level of abstraction for developing messaging applications, it is an industry-standard API (which will hopefully result in support from other messaging products), and the AMI model clearly demarcates MQSeries (or other middleware product) administration from application development. For this reason, AMI separates services and policies from the message itself, with services and policies being administered externally to the messaging application. This is clearly a superior model, and it enhances the adaptability of messaging applications to changing requirements.

AMI currently supports C, C++, and Java. Two APIs for C are provided: a high-level procedural API and a low-level object API. While MQSeries' AMI implementation lacks support for COBOL, I suspect this cannot be far behind its support for C. However, its lack of support for Perl may be a disadvantage.

In this article I used the high-level C API to demonstrate standard messaging applications that send and receive messages and implement request/response processing using asynchronous messaging. The request/response-type processing demonstrates the strength of the AMI – application developers no longer have to worry about how the request/response model is implemented, as they can use the AMI administration tool to change the policy and, hence, many of the message's attributes, including priority, persistence, and wait time for the receiving application.

In summary, the AMI is an exciting development for MQSeries application developers and MQSeries administrators. The AMI delivers on the promise of providing a higher level of abstraction and better application control. It deserves the endorsement of the industry that it has received and it sets a new standard in the development of messaging applications.

Ashish Joshi
Consultant (USA)

A system generator for MQSeries (part 3)

This is the third and concluding part of this article on generating an MQ system automatically (the first part appeared in the November 1999 issue of *MQ Update*).

MQSDEFA

```
* Trigger attributes
         NOTRIGGER +
         TRIGTYPE( FIRST ) +
         TRIGDPTH( 1 ) +
         TRIGMPRI( 0 ) +
         TRIGDATA( ' ' ) +
         PROCESS( ' ' ) +
         INITQ( ' ' )
*
******
DEFINE QLOCAL( 'SYSTEM.ADMIN.PERFM.EVENT' ) +

* Common queue attributes
         DESCR( 'System performance related event queue' ) +
         LIKE( 'SYSTEM.ADMIN.QMGR.EVENT' )
*
******
DEFINE QLOCAL( 'SYSTEM.ADMIN.CHANNEL.EVENT' ) +

* Common queue attributes
         DESCR( 'System channel related event queue' ) +
         LIKE( 'SYSTEM.ADMIN.QMGR.EVENT' )
*
*
*********************************************************************
* NON-SYSTEM DEFINITIONS
*********************************************************************
*
* The queue manager-specific commands define the dead-letter local
* queue for a particular queue manager.
*
* This is a SAMPLE definition of what is needed.
*
* The name of the dead letter queue may be specified in the ALTER
* QMGR command below. While the attributes of the dead-letter queue
* may be changed, if they are changed in such a way that, when the
* queue manager tries to PUT a message on the dead-letter queue,
* the operation fails, the result is the message being discarded.
```

```
*
******
DEFINE QLOCAL( '&SYSID..DEAD.QUEUE' ) +

* Common queue attributes
         DESCR( '&SYSID DEAD-LETTER QUEUE' ) +
         PUT( DISABLED ) +
         DEFPRTY( 0 ) +
         DEFPSIST( YES ) +

* Local queue attributes
         GET( DISABLED ) +
         SHARE +
         DEFSOPT( SHARED ) +
         MSGDLVSQ( FIFO ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 999999999 ) +
         MAXMSGL( 4194304 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'SYSTEM' ) +
         USAGE( NORMAL ) +

* Event control attributes
         QDPMAXEV( ENABLED ) +
         QDPHIEV( DISABLED ) +
         QDEPTHHI( 80 ) +
         QDPLOEV( DISABLED ) +
         QDEPTHLO( 40 ) +
         QSVCIEV( NONE ) +
         QSVCINT( 999999999 ) +

* Trigger attributes
         NOTRIGGER +
         TRIGTYPE( NONE ) +
         TRIGMPRI( 0 ) +
         TRIGDPTH( 1 ) +
         TRIGDATA( ' ' ) +
         PROCESS( ' ' ) +
         INITQ( ' ' )
*
******
* Alter the queue manager attributes for this instance.
*
ALTER QMGR +
         DESCR( '&SYSID ,  MQSERIES FOR MVS/ESA - V1.1.4' ) +
         TRIGINT( 300000 ) +
         MAXHANDS( 256 ) +
         INHIBTEV( ENABLED ) +
         LOCALEV( ENABLED ) +
         REMOTEEV( ENABLED ) +
         STRSTPEV( ENABLED ) +
         PERFMEV( ENABLED ) +
         DEADQ( '&SYSID..DEAD.QUEUE' )
*
*
*********************************************************************
* CICS ADAPTER DEFINITIONS
*********************************************************************
*
* The CICS initiation queue is used by the CKTI CICS transaction.
* This queue is required for communication between CICS and the
* queue manager.
*
* This is a SAMPLE definition of what is needed.
*
* You may change the name of the CICS initiation queue if you wish.
* The name should match the one in the CICS system initialization
* table or the SYSIN override in the statement below.
*
*  INITPARM=(CSQCPARM='IQ=CICS01.INITQ, ...
*
******
DEFINE QLOCAL( '&CICID..INITQ' ) +

* Common queue attributes
         DESCR( 'CKTI initiation queue' ) +
         PUT( ENABLED ) +
         DEFPRTY( 5 ) +
         DEFPSIST( YES ) +

* Local queue attributes
         GET( ENABLED ) +
         SHARE +
         DEFSOPT( EXCL ) +
         MSGDLVSQ( FIFO ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 100 ) +
         MAXMSGL( 1000 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'SYSTEM' ) +
         USAGE( NORMAL ) +

* Event control attributes
         QDPMAXEV( ENABLED ) +
         QDPHIEV( DISABLED ) +
         QDEPTHHI( 80 ) +
         QDPLOEV( DISABLED ) +
         QDEPTHLO( 40 ) +
         QSVCIEV( NONE ) +
         QSVCINT( 999999999 ) +

* Trigger attributes
         NOTRIGGER +
         TRIGTYPE( NONE ) +
         TRIGMPRI( 0 ) +
         TRIGDPTH( 1 ) +
         TRIGDATA( ' ' ) +
         PROCESS( ' ' ) +
         INITQ( ' ' )
*
*
*********************************************************************
* start chinit / listener/ channels
*********************************************************************
START CHINIT PARM(&SYSID.CHIP)
*********************************************************************
* End of CSQ4INP2
*********************************************************************
./  ADD NAME=CSQ4DISX
*********************************************************************

*********************************************************************
*
*                MQSeries for MVS/ESA
* CSQDISX sample for distributed queueing without CICS
*
*********************************************************************
*
* In order to use distributed queueing facilites without CICS,
* the objects in this section of code must be created the first
* time that a queue manager is started. This can be done by
* including this data set in the CSQINP2 DD concatenation in the
* queue manager started task procedure, as shown in the sample
* procedure CSQ4MSTR.
*
* Once the objects are successfully created, there is no need to
* redefine them on subsequent queue manager starts, so this data
* set can be removed from the CSQINP2 DD concatenation. If the data
* set is not removed from CSQINP2, define operations will fail
* with an error message stating that the object already exists. You
* can also add the keyword REPLACE to each command if the definitions
* are to be reset on every start-up.
*
*********************************************************************
*
* The following sample definitions show what is required at the
* sending and receiving ends to send messages. They show a
* sender-receiver channel pair using tcp/ip as the transport.
*
* Changes necessary to use a server-requester channel pair and
* to use LU6.2 as the transport are included.
*
*********************************************************************
* Sending-end definitions
*********************************************************************
*
*********************************************************************
* Common definitions
*********************************************************************
*
* These definitions are required to use
* the distributed queueing facility.
*
******
DEFINE QLOCAL( 'SYSTEM.CHANNEL.SYNCQ' ) +

* COMMON QUEUE ATTRIBUTES
         DESCR( 'SYSTEM CHANNEL SYNCHRONIZATION QUEUE' ) +
         PUT( ENABLED )  +
         DEFPRTY( 5 )    +
         DEFPSIST( YES ) +

* LOCAL QUEUE ATTRIBUTES
         GET( ENABLED ) +
         SHARE +
         DEFSOPT( EXCL ) +
         MSGDLVSQ( FIFO ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 10000 ) +
         MAXMSGL( 4194304 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'SYSTEM' ) +

* EVENT CONTROL ATTRIBUTES
         QDPMAXEV( ENABLED ) +
         QDPHIEV( DISABLED ) +
         QDEPTHHI( 80 ) +
         QDPLOEV( DISABLED ) +
         QDEPTHLO( 40 ) +
         QSVCIEV( NONE ) +
         QSVCINT( 999999999 ) +

* TRIGGER ATTRIBUTES
         NOTRIGGER +
         TRIGTYPE( NONE ) +
         TRIGMPRI( 0 ) +
         TRIGDPTH( 1 ) +
         TRIGDATA( ' ' ) +
         PROCESS( ' ' ) +
         INITQ( ' ' )
*
******
DEFINE QLOCAL( 'SYSTEM.CHANNEL.INITQ' ) +

* COMMON QUEUE ATTRIBUTES
         DESCR( 'SYSTEM CHANNEL INITIATION QUEUE' ) +
         PUT( ENABLED )  +
         DEFPRTY( 5 )    +
         DEFPSIST( YES ) +

* LOCAL QUEUE ATTRIBUTES
         GET( ENABLED ) +
         SHARE +
         DEFSOPT( EXCL ) +
         MSGDLVSQ( FIFO ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 1000 ) +
         MAXMSGL( 4194304 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'SYSTEM' ) +

* EVENT CONTROL ATTRIBUTES
         QDPMAXEV( ENABLED ) +
         QDPHIEV( DISABLED ) +
         QDEPTHHI( 80 ) +
         QDPLOEV( DISABLED ) +
         QDEPTHLO( 40 ) +
         QSVCIEV( NONE ) +
         QSVCINT( 999999999 ) +

* TRIGGER ATTRIBUTES
         NOTRIGGER +
         TRIGTYPE( NONE ) +
         TRIGMPRI( 0 ) +
         TRIGDPTH( 1 ) +
         TRIGDATA( ' ' ) +
         PROCESS( ' ' ) +
         INITQ( ' ' )
*
******
DEFINE QLOCAL( 'SYSTEM.CHANNEL.REPLY.INFO' ) +
* COMMON QUEUE ATTRIBUTES
         DESCR( 'SYSTEM CHANNEL COMMAND REPLY DATA QUEUE' ) +
         PUT( ENABLED )  +
         DEFPRTY( 5 )    +
         DEFPSIST( YES ) +

* LOCAL QUEUE ATTRIBUTES
         GET( ENABLED ) +
         SHARE +
         DEFSOPT( EXCL ) +
         MSGDLVSQ( FIFO ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 1000 ) +
         MAXMSGL( 4194304 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'SYSTEM' ) +

* EVENT CONTROL ATTRIBUTES
         QDPMAXEV( ENABLED ) +
         QDPHIEV( DISABLED ) +
         QDEPTHHI( 80 ) +
         QDPLOEV( DISABLED ) +
         QDEPTHLO( 40 ) +
         QSVCIEV( NONE ) +
         QSVCINT( 999999999 ) +

* TRIGGER ATTRIBUTES
         NOTRIGGER +
         TRIGTYPE( NONE ) +
         TRIGMPRI( 0 ) +
         TRIGDPTH( 1 ) +
         TRIGDATA( ' ' ) +
         PROCESS( ' ' ) +
         INITQ( ' ' )
*
*
*******************************************************************
* END OF CSQ4DISX
*******************************************************************
./  ADD NAME=CSQ4STGC
*******************************************************************
*                                                             *
*
*******************************************************************
*
*                MQSeries for MVS/ESA
* CSQ4STGC sample for storage class definitions
*
```

```
*********************************************************************
*
* This sample contains a set of storage class definitions that
* associate storage classes with page sets. They may be invoked from
* the CSQINP2 concatenation every time the queue manager is started.
*
* The storage class definitions below are required.
*    SYSTEM
*    REMOTE
*    DEFAULT
*    NODEFINE
*
* It is recommended that storage classes REMOTE, DEFAULT, and
* NODEFINE are not defined as mapping to page set 00 (this sample
* shows them mapped to page set 01) so as to keep user and system
* messages on separate page sets. Compliance with this standard
* requires that at least two page data sets are defined.
*
* Further storage class definitions should be added to this sample
* as required.
*
*********************************************************************
*
*********************************************************************
* STGCLASS definitions - please change them for your own needs
*********************************************************************
*
* Associate storage classes with page sets.

DEFINE STGCLASS( 'SYSTEM' ) +
       PSID( 00 )

DEFINE STGCLASS( 'SYSTEMST' ) +
       DESCR( ' DEFINE STORAGE CLASS WITH DEFAULTS' ) +
       XCFGNAME( ' ' ) +
       XCFMNAME( ' ' ) +
       PSID( 01 )

DEFINE STGCLASS( 'REMOTE' ) +
       PSID( 01 )

DEFINE STGCLASS( 'NODEFINE' ) +
       PSID( 01 )

DEFINE STGCLASS( 'STGRABFB' ) +
       PSID( 02 )

DEFINE STGCLASS( 'STGRABTG' ) +
       PSID( 03 )
```

```
DEFINE STGCLASS( 'STGAABFB' ) +
       PSID( 04 )

DEFINE STGCLASS( 'STGAABTG' ) +
       PSID( 05 )

DEFINE STGCLASS( 'STGINGFB' ) +
       PSID( 06 )

DEFINE STGCLASS( 'STGINGTG' ) +
       PSID( 07 )

DEFINE STGCLASS( 'STGRSTFB' ) +
       PSID( 08 )

DEFINE STGCLASS( 'STGRSTTG' ) +
       PSID( 09 )

*
**********************************************************************
* End of CSQ4STGC
**********************************************************************
./  ADD NAME=CSQINPX
**********************************************************************
*                                                                    *
**********************************************************************
*
*                        MQSeries for MVS/ESA
* CSQINPX sample
*
**********************************************************************
*
* This sample data set contains an example of a set of commands
* that could be issued whenever distributed queuing without CICS
* is started.
*
**********************************************************************
* Start Listeners
**********************************************************************
*
* You must start a listener for each communication protocol you use.
*
******
START LISTENER TRPTYPE( LU62 ) LUNAME( LIST&SYSID )
*
*
**********************************************************************
* Start and stop channels
**********************************************************************
*
```

```
* The sender channels normally start automatically when a trigger
* message is put on the channel initiation queue. Similarly,
* receiver and server channels start automatically when a message
* is received from a remote queue manager. They stop automatically
* when there is no more work for them.
*
* It is necessary to restart channels manually only when they
* stopped as the result of an error or were stopped manually.
* Both these conditions are cleared when the channel initiator
* is started, so there is no need to issue any START CHANNEL
* commands.
*
* However, if you don't want some channels to start manually,
* then issue STOP commands for them when the channel initiator
* starts. Later, when you want them to start, you will have to issue
* START commands for them.
*
*********************************************************************
* Display channel status
*********************************************************************
*
* Show the status of all the channels that are currently defined.
*
******
DISPLAY CHSTATUS(*) CURRENT ALL

*
*********************************************************************
* End of CSQ4INPX
*********************************************************************
./  ENDUP
/*
```

MQSDEFB

```
//&USERID.J JOB (,EXP),'&USERID',
// NOTIFY=&USERID,
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
// TIME=3
//*
/*ROUTE  XEQ &LPAR
//*
//* THIS JOB CREATES THE MQSERIES BDSDS AND LOGCOPIES
//*
//*****************************************************************/
//* JOB NAME = MQSDEFB                                             */
//*                                                                */
//* (C) COPYRIGHT INTERPAY 1999                                    */
//*                                                                */
//* STATUS   = VERSION 1                                             */
//*                                                                  */
//* FUNCTION = DEFINE MQSERIES DATASETS                              */
//*                                                                  */
//* NOTES                                                            */
//*   1. THE INSTALL CLIST SHOULD DO MOST OF THE WORK, THOUGH THE  */
//*      INSTALLER MAY INSPECT THE AMS COMMANDS AND JCL AND EDIT   */
//*      THEM WHERE NECESSARY TO SUIT YOUR SITE'S REQUIREMENTS.    */
//*   2. DSNAMES, CONTROL INTERVAL SIZE, RECORDSIZE, LINEAR,        */
//*      NONORDERED, AND SHAREOPTIONS MUST NOT BE CHANGED FOR       */
//*      DIRECTORY AND CATALOGUE DATA.                              */
//*   3. MANY PARAMETERS DO NOT APPLY TO DIRECTORY AND CATALOGUE   */
//*      DATA, INCLUDING SPANNED, EXCEPTIONEXIT, SPEED,             */
//*      BUFFERSPACE, AND WRITECHECK.                               */
//*   4. DATA SET SIZES, PASSWORDS, AND VOLUMES MAY BE CHANGED.    */
//*      MSS STAGING OPTIONS MAY BE ADDED.                          */
//*                                                                  */
//CSQLOG2 EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN    DD  *
   DEFINE CLUSTER                       -
          (NAME(&SYSID..BSDS01)         -
           VOLUMES(&BSDS1VOL)           -
           REUSE                        -
           SHAREOPTIONS(2 3) )          -
       DATA                             -
          (NAME(&SYSID..BSDS01.DATA)    -
           RECORDS(200 100)             -
           RECORDSIZE(4089 4089)        -
           CONTROLINTERVALSIZE(4096)    -
           FREESPACE(0 20)              -
           KEYS(4 0) )                  -
      INDEX                             -
         (NAME(&SYSID..BSDS01.INDEX)    -
          RECORDS(5 5)                  -
          CONTROLINTERVALSIZE(1024) )

   DEFINE CLUSTER                       -
          (NAME(&SYSID..BSDS02)         -
           VOLUMES(&BSDS2VOL)           -
           REUSE                        -
           SHAREOPTIONS(2 3) )          -
       DATA                             -
          (NAME(&SYSID..BSDS02.DATA)    -
           RECORDS(200 100)             -
           RECORDSIZE(4089 4089)        -
           CONTROLINTERVALSIZE(4096)    -
           FREESPACE(0 20)              -
           KEYS(4 0) )                  -
      INDEX                             -
```

```
       (NAME(&SYSID..BSDS02.INDEX)    -
        RECORDS(5 5)                  -
        CONTROLINTERVALSIZE(1024) )

DEFINE CLUSTER -
       (NAME (&SYSID..LOGCOPY1.DS01)    -
        LINEAR                          -
        REUSE                           -
        VOLUMES(&LOG11VOL)              -
        CYLINDERS(&CYLS) )              -
    DATA                                -
       (NAME(&SYSID..LOGCOPY1.DS01.DATA) )

DEFINE CLUSTER -
       (NAME (&SYSID..LOGCOPY1.DS02)    -
        LINEAR                          -
        REUSE                           -
        VOLUMES(&LOG12VOL)              -
        CYLINDERS(&CYLS) )              -
    DATA                                -
       (NAME(&SYSID..LOGCOPY1.DS02.DATA) )

DEFINE CLUSTER -
       (NAME (&SYSID..LOGCOPY1.DS03)    -
        LINEAR                          -
        REUSE                           -
        VOLUMES(&LOG13VOL)              -
        CYLINDERS(&CYLS) )              -
    DATA                                -
       (NAME(&SYSID..LOGCOPY1.DS03.DATA) )

DEFINE CLUSTER -
       (NAME (&SYSID..LOGCOPY2.DS01)    -
        LINEAR                          -
        REUSE                           -
        VOLUMES(&LOG21VOL)              -
        CYLINDERS(&CYLS) )              -
    DATA                                -
       (NAME(&SYSID..LOGCOPY2.DS01.DATA) )

DEFINE CLUSTER -
       (NAME (&SYSID..LOGCOPY2.DS02)    -
        LINEAR                          -
        REUSE                           -
        VOLUMES(&LOG22VOL)              -
        CYLINDERS(&CYLS) )              -
    DATA                                -
       (NAME(&SYSID..LOGCOPY2.DS02.DATA) )

DEFINE CLUSTER -
```

```
         (NAME (&SYSID..LOGCOPY2.DS03)    -
          LINEAR                          -
          REUSE                           -
          VOLUMES(&LOG23VOL)              -
          CYLINDERS(&CYLS) )              -
      DATA                                -
         (NAME(&SYSID..LOGCOPY2.DS03.DATA) )

/*
//CSQTLOG EXEC PGM=CSQJU003
//STEPLIB  DD  DISP=SHR,DSN=MQM.SCSQAUTH
//SYSUT1   DD  DISP=OLD,DSN=&SYSID..BSDS01
//SYSUT2   DD  DISP=OLD,DSN=&SYSID..BSDS02
//SYSPRINT DD  SYSOUT=*,DCB=BLKSIZE=629
//SYSIN    DD  *
 NEWLOG DSNAME=&SYSID..LOGCOPY1.DS01,COPY1
 NEWLOG DSNAME=&SYSID..LOGCOPY1.DS02,COPY1
 NEWLOG DSNAME=&SYSID..LOGCOPY1.DS03,COPY1
 NEWLOG DSNAME=&SYSID..LOGCOPY2.DS01,COPY2
 NEWLOG DSNAME=&SYSID..LOGCOPY2.DS02,COPY2
 NEWLOG DSNAME=&SYSID..LOGCOPY2.DS03,COPY2
/*
//
```

MQSDEFC

```
//&USERID.J JOB (,EXP),'&USERID',
// NOTIFY=&USERID,
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
// TIME=3
//*
/*ROUTE  XEQ &LPAR
//*
//*****************************************************************/
//* JOB NAME = MQSDEFC                                            */
//*                                                               */
//* DESCRIPTIVE NAME = INSTALLATION JOB STREAM                    */
//*                                                               */
//* LICENSED MATERIALS - PROPERTY OF INTERPAY                     */
//*                                                               */
//* (C) COPYRIGHT INTERPAY 1999                                   */
//*                                                               */
//* STATUS   = VERSION 1                                          */
//*                                                               */
//* FUNCTION = DEFINE MQSERIES DEFAULT BATCH/TSO ADAPTER          */
//*                                                               */
//*                                                               */
//* NOTES                                                         */
//*   1. THE INSTALL CLIST SHOULD DO MOST OF THE WORK, THOUGH THE */
//*      INSTALLER MAY INSPECT THE AMS COMMANDS AND JCL AND EDIT   */
//*      THEM WHERE NECESSARY TO SUIT YOUR SITE'S REQUIREMENTS.    */
//*   2. DSNAMES, CONTROL INTERVAL SIZE, RECORDSIZE, LINEAR,       */
//*      NONORDERED, AND SHAREOPTIONS MUST NOT BE CHANGED FOR      */
//*      DIRECTORY AND CATALOGUE DATA.                             */
//*   3. MANY PARAMETERS DO NOT APPLY TO DIRECTORY AND CATALOGUE   */
//*      DATA, INCLUDING SPANNED, EXCEPTIONEXIT, SPEED,            */
//*      BUFFERSPACE, AND WRITECHECK.                              */
//*   4. DATA SET SIZES, PASSWORDS, AND VOLUMES MAY BE CHANGED.    */
//*      MSS STAGING OPTIONS MAY BE ADDED.                         */
//*                                                                */
//****************************************************************/
//DEFLIBS  EXEC PGM=IEFBR14
//DD1      DD DISP=(NEW,CATLG,DELETE),
// DSN=&SYSID..PERM.LOAD,
// SPACE=(CYL,(1,1,20)),UNIT=SYSDA,VOL=SER=&VOL,
// DCB=(BLKSIZE=32760,RECFM=U,DSORG=PO)
//*                                                                */
//ASM    EXEC PGM=IEV90,
//       REGION=1024K,
//       PARM='DECK,NOOBJECT,LIST'
//SYSLIB   DD DSN=SYS1.MACLIB,DISP=SHR
//****************************************************************
//         DD DSN=MQM.SCSQMACS,DISP=SHR
//         DD DSN=&SYSID..SCSQPROC,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(1700,(400,400))
//SYSUT2   DD UNIT=SYSDA,SPACE=(1700,(400,400))
//SYSUT3   DD UNIT=SYSDA,SPACE=(1700,(400,400))
//SYSPUNCH DD DSN=&LOAD,
//       UNIT=SYSDA,DISP=(,PASS),
//       SPACE=(400,(100,100,1))
//****************************************************************
//* CHANGE USERSRC.ASSEMBLE TO THE DATASET CONTAINING SOURCE
//****************************************************************
//SYSIN    DD DSN=&SYSID..SCSQPROC(CSQBDEFV),DISP=SHR
//SYSPRINT DD SYSOUT=*
//*
//****************************************************************
//* LINK-EDIT
//****************************************************************
//LKED   EXEC PGM=IEWL,REGION=1024K,
//       PARM=(LIST,LET,XREF,RENT),COND=(7,LT,ASM)
//****************************************************************
//SYSOBJ   DD DSN=MQM.SCSQLOAD,DISP=SHR
//****************************************************************
//* CHANGE USER.LOADLIB TO THE LIBRARY THAT WILL CONTAIN THE
//* TRANAEXT LOAD MODULE. THIS LIBRARY NEEDs TO BE CONCATENATED
//* TO DD CSQXLIB IN THE CHANNEL INITIATOR JCL.
//****************************************************************
//SYSLMOD  DD DSN=&SYSID..PERM.LOAD,DISP=(SHR,KEEP)
```

```
//SYSUT1   DD UNIT=SYSDA,DCB=BLKSIZE=1024,
//       SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=*
//SYSLIN   DD DSN=&LOAD,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
//*******************************************************************
//* NOTE: CSQBSTUB HAS BEEN INCLUDED IN THE LOAD MODULE, BUT IS
//* NECESSARY ONLY IF THE EXIT WILL ISSUE MQI CALLS.
//*******************************************************************
//SYSIN DD *
 INCLUDE SYSOBJ(CSQBSTUB)
 NAME  CSQBDEFV(R)
/*
```

MQSDEFP

```
//&USERID.J JOB (,EXP),'&USERID',
// NOTIFY=&USERID,
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
// TIME=3
//*
/*ROUTE  XEQ &LPAR
//*
//* THIS JOB CREATES THE MQSERIES PAGESETS
//*
//*******************************************************************/
//* JOB NAME = MQSDEFP                                              */
//*                                                                 */
//* DESCRIPTIVE NAME = INSTALLATION JOB STREAM                      */
//*                                                                 */
//* LICENSED MATERIALS - PROPERTY OF INTERPAY                       */
//*                                                                 */
//* (C) COPYRIGHT INTERPAY 1999                                     */
//*                                                                 */
//* STATUS   = VERSION 1                                            */
//*                                                                 */
//* FUNCTION = DEFINE MQSERIES DATASETS                             */
//*                                                                 */
//* NOTES                                                           */
//*   1. THE INSTALL CLIST SHOULD DO MOST OF THE WORK, THOUGH THE  */
//*      INSTALLER MAY INSPECT THE AMS COMMANDS AND JCL AND EDIT   */
//*      THEM WHERE NECESSARY TO SUIT YOUR SITE'S REQUIREMENTS.    */
//*   2. DSNAMES, CONTROL INTERVAL SIZE, RECORDSIZE, LINEAR,       */
//*      NONORDERED, AND SHAREOPTIONS MUST NOT BE CHANGED FOR      */
//*      DIRECTORY AND CATALOGUE DATA.                              */
//*   3. MANY PARAMETERS DO NOT APPLY TO DIRECTORY AND CATALOGUE   */
//*      DATA, INCLUDING SPANNED, EXCEPTIONEXIT, SPEED,            */
//*      BUFFERSPACE, AND WRITECHECK.                               */
//*   4. DATA SET SIZES, PASSWORDS, AND VOLUMES MAY BE CHANGED.    */
//*      MSS STAGING OPTIONS MAY BE ADDED.                        */
//*                                                               */
//DEFINE   EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *

   DEFINE CLUSTER                         -
          (NAME(&SYSID..PSID00)           -
           RECORDS(2000 1000)             -
           LINEAR                         -
           VOLUMES(&VOL1PS)               -
           SHAREOPTIONS(2 3) )            -
       DATA                               -
          (NAME(&SYSID..PSID00.DATA) )

DEFINE CLUSTER                         -
          (NAME(&SYSID..PSID01)           -
           RECORDS(3000 1000)             -
           LINEAR                         -
           VOLUMES(&VOL2PS)               -
           SHAREOPTIONS(2 3) )            -
       DATA                               -
          (NAME(&SYSID..PSID01.DATA) )

   DEFINE CLUSTER                         -
          (NAME(&SYSID..PSID02)           -
           RECORDS(3000 1000)             -
           LINEAR                         -
           VOLUMES(&VOL1PS)               -
           SHAREOPTIONS(2 3) )            -
       DATA                               -
          (NAME(&SYSID..PSID02.DATA) )

   DEFINE CLUSTER                         -
          (NAME(&SYSID..PSID03)           -
           RECORDS(3000 1000)             -
           LINEAR                         -
           VOLUMES(&VOL2PS)               -
           SHAREOPTIONS(2 3) )            -
       DATA                               -
          (NAME(&SYSID..PSID03.DATA) )

   DEFINE CLUSTER                         -
          (NAME(&SYSID..PSID04)           -
           RECORDS(3000 1000)             -
           LINEAR                         -
           VOLUMES(&VOL1PS)               -
           SHAREOPTIONS(2 3) )            -
       DATA                               -
          (NAME(&SYSID..PSID04.DATA) )
```

```
DEFINE CLUSTER                       -
       (NAME(&SYSID..PSID05)         -
        RECORDS(3000 1000)           -
        LINEAR                       -
        VOLUMES(&VOL2PS)             -
        SHAREOPTIONS(2 3) )          -
    DATA                             -
       (NAME(&SYSID..PSID05.DATA) )

DEFINE CLUSTER                       -
       (NAME(&SYSID..PSID06)         -
        RECORDS(3000 1000)           -
        LINEAR                       -
        VOLUMES(&VOL1PS)             -
        SHAREOPTIONS(2 3) )          -
    DATA                             -
       (NAME(&SYSID..PSID06.DATA) )

DEFINE CLUSTER                       -
       (NAME(&SYSID..PSID07)         -
        RECORDS(3000 1000)           -
        LINEAR                       -
        VOLUMES(&VOL2PS)             -
        SHAREOPTIONS(2 3) )          -
    DATA                             -
       (NAME(&SYSID..PSID07.DATA) )

DEFINE CLUSTER                       -
       (NAME(&SYSID..PSID08)         -
        RECORDS(3000 1000)           -
        LINEAR                       -
        VOLUMES(&VOL1PS)             -
        SHAREOPTIONS(2 3) )          -
    DATA                             -
       (NAME(&SYSID..PSID08.DATA) )

DEFINE CLUSTER                       -
       (NAME(&SYSID..PSID09)         -
        RECORDS(3000 1000)           -
        LINEAR                       -
        VOLUMES(&VOL2PS)             -
        SHAREOPTIONS(2 3) )          -
    DATA                             -
       (NAME(&SYSID..PSID09.DATA) )

DEFINE CLUSTER                       -
       (NAME(&SYSID..PSID10)         -
        RECORDS(3000 1000)           -
        LINEAR                       -
```

```
            VOLUMES(&VOL1PS)            -
            SHAREOPTIONS(2 3) )         -
        DATA                            -
           (NAME(&SYSID..PSID10.DATA) )

/*
//FORM     EXEC  PGM=CSQUTIL
//STEPLIB  DD    DISP=SHR,DSN=MQM.SCSQAUTH
//CSQP0000 DD    DISP=OLD,DSN=&SYSID..PSID00
//CSQP0001 DD    DISP=OLD,DSN=&SYSID..PSID01
//CSQP0002 DD    DISP=OLD,DSN=&SYSID..PSID02
//CSQP0003 DD    DISP=OLD,DSN=&SYSID..PSID03
//CSQP0004 DD    DISP=OLD,DSN=&SYSID..PSID04
//CSQP0005 DD    DISP=OLD,DSN=&SYSID..PSID05
//CSQP0006 DD    DISP=OLD,DSN=&SYSID..PSID06
//CSQP0007 DD    DISP=OLD,DSN=&SYSID..PSID07
//CSQP0008 DD    DISP=OLD,DSN=&SYSID..PSID08
//CSQP0009 DD    DISP=OLD,DSN=&SYSID..PSID09
//CSQP0010 DD    DISP=OLD,DSN=&SYSID..PSID10
//SYSPRINT DD    SYSOUT=*
//SYSIN    DD    *
FORMAT
/*
//
```

MQSDEFS

```
//&USERID.P JOB    (ACCT#),'INSTALL',CLASS=A,MSGCLASS=X,
//          NOTIFY=&USERID
//*****************************************************************/
//* JOB NAME = MQSERIES PROCEDURES                                */
//*                                                               */
//* DESCRIPTIVE NAME = INSTALLATION JOB STREAM                    */
//*                                                               */
//* FUNCTION  = MVS MODIFICATIONS                                 */
//*                                                               */
//* PSEUDOCODE =                                                  */
//*   MQIPM    STEP      FOR UPDATING THE MVS PROCLIB WITH CICS:  */
//*                      1) STARTUP PROCEDURES                    */
//*                                                               */
//* NOTES =                                                       */
//*   PLEASE CHECK THIS JOB CAREFULLY TO ENSURE THAT THE SYSTEM   */
//*   LIBRARY NAMES ARE THE CORRECT ONES FOR YOUR SITE.           */
//***********************************************
//*    ADD CATALOGED PROCEDURES TO PROCLIB        *
//***********************************************
/*ROUTE  XEQ &LPAR
//***********************************************
//MQIPM EXEC PGM=IEBUPDTE,PARM=NEW
```

```
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT2   DD DISP=SHR,DSN=SYS1.PROCLIB.DB2  < or another proclib
//SYSIN    DD DATA
./  ADD NAME=&SYSID.MSTR
//&SYSID.MSTR EXEC PGM=CSQYASCP,REGION=7168K
//STEPLIB   DD DSN=MQM.SCSQANLE,DISP=SHR
//          DD DSN=MQMP.SMQMEXIT,DISP=SHR
//BSDS1     DD DSN=&SYSID..BSDS01,DISP=SHR
//BSDS2     DD DSN=&SYSID..BSDS02,DISP=SHR
//CSQINP1   DD DSN=&SYSID..SCSQPROC(CSQ4INP1),DISP=SHR
//CSQINP2   DD DSN=&SYSID..SCSQPROC(CSQ4STGC),DISP=SHR
//          DD DSN=&SYSID..SCSQPROC(CSQ4DISX),DISP=SHR
//          DD DSN=&SYSID..SCSQPROC(CSQ4INP2),DISP=SHR
//CSQOUT1   DD SYSOUT=X
//CSQOUT2   DD SYSOUT=X
//CSQP0000  DD DSN=&SYSID..PSID00,DISP=SHR
//CSQP0001  DD DSN=&SYSID..PSID01,DISP=SHR
//CSQP0002  DD DSN=&SYSID..PSID02,DISP=SHR
//CSQP0003  DD DSN=&SYSID..PSID03,DISP=SHR
//CSQP0004  DD DSN=&SYSID..PSID04,DISP=SHR
//CSQP0005  DD DSN=&SYSID..PSID05,DISP=SHR
//CSQP0006  DD DSN=&SYSID..PSID06,DISP=SHR
//CSQP0007  DD DSN=&SYSID..PSID07,DISP=SHR
//CSQP0008  DD DSN=&SYSID..PSID08,DISP=SHR
//CSQP0009  DD DSN=&SYSID..PSID09,DISP=SHR
//CSQP0010  DD DSN=&SYSID..PSID10,DISP=SHR
//*
./  ADD NAME=&SYSID.CHIN
//&SYSID.CHIN EXEC PGM=CSQXJST,REGION=7168K,TIME=1440
//STEPLIB   DD DSN=MQM.SCSQANLE,DISP=SHR
//          DD DSN=MQMP.SMQMEXIT,DISP=SHR
//          DD DSN=EDC.SEDCLINK,DISP=SHR
//CSQXLIB   DD DSN=MQMP.SMQMEXIT,DISP=SHR
//CSQINPX   DD DSN=&SYSID..SCSQPROC(CSQINPX),DISP=SHR
//CSQOUTX   DD SYSOUT=X
./  ENDUP
```

LOCAL CUSTOMIZATION

It may be necessary for you to make some minor changes to the system generator tool to accommodate such factors as local naming conventions.

In particular, you need to consider the following:

- The CICS system name (MQSDEFA skeleton) for initiation queues, which is also used in the CSQCPARM (INITPARM) for your CICS region.

- The ‘proclib’ name (MQSDEFS), which is for the SYS1.PROCLIB.DB2 environment in our example, though this could be for a different environment in your case.
- The QMANAGER trigger interval, which is set to 300,000 (instead of the default 999,999,999).
- The MSTR and CHIN procedure expect the parameter modules in a library called MQMP.SMQMEXIT (this should be APF-authorized).
- You have to create the parameter modules separately from the generator tool.
- CSQ4INP2 (which is a member of SCSQPROC) contains a START statement for the channel initiator address space. This expects a parameter module with the name &SYSIDCHIP (for instance, CSQ4CHIP).
- The ‘stgclass’ names should be changed to suit your requirements.

Paul Jansen
Systems Programmer
Interpay (The Netherlands)

MQSeries for MVS – a quick primer

This article is intended as a ‘quick primer’ for those starting with MQSeries on MVS.

THE BASICS

MQSeries lets MVS applications use message queueing to participate in message-driven processing. Applications can communicate across different platforms using the appropriate message queueing software products, such as MQSeries for MVS/ESA and MQSeries for OS/2.

MQSeries products use a common application programming interface, the MQI, on whatever platforms they run. This means that the calls made by applications and the messages they exchange are common between platforms.

Applications can be created in:

- MVS’s batch and TSO environment
- CICS’s transaction environment
- IMS’s transaction environment.

A *queue manager* provides queueing services to applications. More than one queue manager (subsystem) may run within each MQSeries system. Distributed queueing supports communication between queue managers. Either CICS intersystem communication or native MVS communication without CICS can be used for distributed queueing.

Both types of queueing can be enabled and used simultaneously on the same instance of MQSeries. However, neither has any knowledge of each other’s channels and, therefore, it’s necessary to ensure that the queue names they use are distinct.

The Client Attachment feature is a component of the MQSeries product that facilitates Distributed Queue Management (DQM). An MQSeries application can be run on an MQSeries client that interacts with one or more MQSeries servers and connects to their queue managers by means of a communications protocol. This feature is free with MQSeries for MVS, so, when installed on a mainframe-based MQSeries server using the communications set-up described above, multiple MQSeries installs and their associated licence cost at the client end (such as NT or OS/2) are saved.

QUEUES

Various types of queue are available in MQSeries, and queues belong to queue managers that maintain them. Queues can be local or remote, and messages are sent across different queue managers using the distributed queueing facility, with communication with MVS being handled either by CICS, TCP/IP, or LU 6.2.

TYPES OF QUEUE

System default queues

These are a set of queue definitions supplied with MQSeries.

Local queue

A local queue is used for processing messages within the same platform on which the queue manager is running.

Event queues

A local or remote queue used to hold messages.

Transmission queue

This is a local queue that has 'usage attribute' *XMIT*. This type of queue should be associated with a sender channel. Any messages flowing from the queue manager through such a channel are essentially taken from the associated transmission queue.

A queue manager can include a default transmission queue. When the queue manager is not able to resolve the channel to be used for a message addressed to a remote queue manager in the network, it puts the message on the default transmission queue, so the message is then moved to the next queue manager, where its address may be resolved.

Initiation queue

A local queue can be used as an initiation queue by a triggered local queue. If a local queue definition states that the queue can trigger a process under certain conditions, a trigger message is written to the initiation queue specified when the conditions arise. The long running trigger monitor task reads this message and the appropriate queue manager process is triggered.

Dead letter queue

A queue manager in a distributed environment should include a dead letter queue. When the queue manager is unable to process a message for whatever reason, it retains the message in the dead letter queue, which prevents channels from coming to a halt.

Remote queue

A remote queue is used for forwarding a message to another queue manager, which should be available on the network using the distributed queueing facility.

Model queue

A queue used as a template for creating dynamic queues. These queues cannot be accessed by any application for putting or getting messages.

Alias queue

Applications connect to either a local or remote queue via this queue.

Dynamic queue

A queue created for use as a 'scratch pad' by an application and deleted when no longer required. Dynamic queues are created using the template from a model queue.

There are two types of dynamic queue.

- *Temporary dynamic queue*
  A queue that will always get deleted when the application that created it issues an *MQCLOSE*.
- *Permanent dynamic queue*
  A queue that can remain in the queue manager even after an application has finished.

System-command input queue

A local queue to which suitably authorized applications can send MQSeries commands. These commands are then retrieved by the Command Server.

CHANNELS

A channel is an MQSeries object that is used to move messages from one queue manager to another or from a client queue manager to a server queue manager. Channels must be in a 'running' state for messages to be transmitted across the network. Channels use the base networking support provided by the APPC or TCP/IP.

The channel initiator is an independent process that enables an MQSeries queue manager to communicate with the network. When successfully started, it allows all the channels to start and makes the network support available on demand.

A sender channel

This should be defined with exactly the same name as a receiver channel on the other queue manager.

A receiver channel

This should be defined as a receiver channel with exactly the same name as a sender channel on the other queue manager.

Channel sequence number queue

A queue that is used to number sequential messages for the unit of work processed when MQSeries subsystems are communicating.

Channel command queue

A queue used to handle commands for channels.

Channel initiation queue

A queue that sends commands to channels.

Channel synchronization queue

A queue that is used to number sequential messages for the unit of work information handled when MQSeries subsystems are communicating.

Channel reply information queue

A queue that handles replies from channel commands.

COMMAND SERVER AND COMMAND PROCESSOR

Messages are read and verified by the command server and the valid ones are passed to the command processor. The command processor processes message, putting replies on to a reply-to queue.

LISTENERS

This process is also started separately for receiving messages coming on the network that are addressed to an MQSeries queue manager. Separate listener processes need to be started for LU62 and TCP/IP.

Saida Davies (UK) © Xephon 2000

MQSeries SupportPacs

IBM has released new MQSeries SupportPacs for download from *http://www.ibm.com/software/mqseries/txppacs*. The SupportPacs, which contain MQSeries client components, are:

- MAC6: MQSeries V5.1 Client for AIX
- MAC7: MQSeries V5.1 Client for HP-UX V10
- MAC8: MQSeries V5.1 Client for HP-UX V11
- MAC9: MQSeries V5.1 Client for Sun Solaris.

One problem when installing SupportPac MA85 for Sun Solaris on a Solaris 2.6 host with MQSeries 5.0 and CSD03 comes when you issue instruction:

```
perl Makefile.PL
```

which results in error message:

```
Perl must be recompiled with '-lthread -lc', and '-D_REENTRANT'
➤  on Solaris
```

The solution to this problem is to upgrade to MQSeries Perl API 1.06, which is now available from *http://www.perl.com/cpan*.

To navigate to it, follow the sequence: *Modules, Modules by name, MQSeries*.

© Xephon 2000

MQ news

New Era of Networks has announced NEONadapter for XML, which allows NEON Integration Servers to integrate XML business-to-business systems with other applications in their enterprise. The software can be used to integrate systems using different XML dialects, such as Microsoft's BizTalk and RosettaNet. It features the ability to parse XML document type definitions and generate formats for the NEONFormatter. It also allows users to add prefixes to the DTD format. A choice of connectivity options is available, allowing any new application that's queue-enabled to work with the adapter. The version for Solaris is out now and supports IBM's MQSeries Integrator for NT and Unix as well as NEON's MQIntegrator for MVS. Availability on NT, AIX, and HP-UX is planned for late 1999. Prices start at US$65,000.

For further information contact:
NEON, 7400 East Orchard Road, Englewood, CO 80111, USA
Tel: +1 303 694 3933
Fax: +1 303 694 3885
Web: http://www.neonsoft.com

New Era of Networks Ltd, Aldermary House, 15 Queen Street, London EC4N 1TX, UK
Tel: + 44 171 329 4669
Fax:+ 44 171 329 4567

* * *

Progress Software has announced SonicMQ, a Java-based Internet messaging server. The software is designed to simplify the development of distributed Internet applications and integrates with the company's own Apptivity Java application server and development suite. SonicMQ uses the new Java Message Service (JMS) standard that's also supported by MQSeries 5. This is a messaging standard that enables Java developers to add enterprise messaging capabilities to their applications.

Progress is to ship both a Developer and Enterprise Edition of SonicMQ, starting November. The Developer Edition is free and the price of the Enterprise Edition is available on request.

For further information contact:
Progress Software, 14 Oak Park, Bedford MA 01730, USA
Tel: +1 781 280 4000
Fax: +1 781 280 4095
Web: http://www.progress.com

Progress Software, The Square, Basingview, Basingstoke, Hants RG21 2EQ, UK
Tel: +44 1256 816668
Fax: +44 1256 463226

* * *

IBM has announced that the launch of MQSeries Integrator V2 has been delayed in order to 'address customer requirements' (the lack of supported platforms and some missing functionality). The planned availability date was 10 December and is now 31 March.