3 MQ update

September 1999

In this issue

Published by

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

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

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.

Printed in England.

PCF programming in Java

INTRODUCTION

As many people familiar with MQSeries will know, PCF is the product's command interface, which enables the development of systems management software. It uses a machine-oriented format and was originally implemented in the C language.

Anyone who's had experience of using the MQSC (**runmqsc**) interface has used PCF, as this is what the **runmqsc** program sends to queue managers. Any action that is possible via the MQSC interface is also possible via PCF. In fact, PCF goes beyond what is available via MQSC, and has additional features that are not implemented in MQSC. Another advantage of PCF is that it also tends to report error conditions a little better. The main drawbacks of PCF are its (original) rather static C implementation and the lack of support for PCF in either MVS or OS/390.

I recently questioned IBM about the lack of OS/390 support at the MQSeries user group. While they acknowledged that this is a long-standing omission from the product, they did not give any information about the possible future inclusion of this feature. Fortunately, a PCF bridge for OS/390 has been developed by the New York-based systems management company, Nastel. I have not had any experience of using this product, though it sounds promising.

In this article I discuss the use of PCF in a systems management strategy, focusing on the newer Java implementation of PCF, and using our own implementation of it in a simple systems management framework as an example.

SYSTEMS MANAGEMENT

I think that most people with operational experience of MQSeries will acknowledge that keeping the whole infrastructure up and running is something of a time consuming task. This is often aggravated by the use of MQSeries in on-line systems that require MQ channels to run

continuously. Much of the administrative overhead of this stems from MQSeries' assured delivery features and the impact they have on channels.

For example, if a transient break in communications has taken place, resulting in a failure to 'handshake' across platforms, MQ channel agents can get into a number of non-functional states. Such states include 'in-doubt messaging' and message sequence errors. The purpose of these states is to ensure that messages are not lost as a result of network errors. While this is often a useful feature, many on-line systems could benefit from 'datagram' style messaging, which is about sending as many messages as possible, not worrying about losing a few.

Another operational MQSeries concern is the notification of failures. It is possible for a 'fire and forget' application to send thousands of messages to a queue manager without their being forwarded to their target queue manager. Processes need to be implemented to handle both the detection of errors and their correction.

It is not generally viable to monitor and correct MQSeries problems manually. Sites that run MQSeries must, therefore, decide whether to purchase a third-party management solution or produce their own tools. The recent release of the Java PCF support pack (it was released in the last six months) has made in-house development a far more attractive proposition.

PCF PROGRAMMING IN JAVA

Java is an excellent language for MQSeries development. Quite apart from the usual arguments about object-orientation and Java's relative ease of use, Java's cross platform nature makes it an ideal partner for cross platform middleware like MQSeries.

Since PCF commands are little more than integer and character data streams, it has always been possible from a technical point of view to issue them from Java. I myself experimented with doing just this before the release of the Java support pack. I found that using Java and (more importantly) object-oriented wrappers increases the usability of PCF significantly.

The IBM support pack contains three main components (see *www.software.ibm.com/ts/mqseries/txppacs/ms0b.html*):

1 A high-level agent that manages connections to MQSeries and also sends and receives messages to and from the command queues.

2 Java classes that map to the five PCF data structures, extending the abstract class *PCFParameter*:

   – *MQCFH*, the PCF header (for return codes etc).

   – *MQCFIN*, for integer parameters.

   – *MQCFIL*, for arrays of integer parameters.

   – *MQCFST*, for string based parameters.

   – *MQCFSL*, for arrays of string parameters.

3 The constants that must be used in PCF commands. These should be placed in interfaces and may, therefore, be implemented by your own code.

Using the IBM classes, a sample piece of code to start a named channel could be as follows:

```
/**
* This method starts the channel specified.
*/
  public void startChannel(PCFAgent agent, String channelName) {

    PCFParameter [] iParameters = new PCFParameter [] {
      new MQCFST (MQCACH_CHANNEL_NAME, channelName),
      };

    try {
      MQMessage [] pcfResponses = agent.send
         (MQCMD_START_CHANNEL, iParameters);

      } catch (MQException ex) {
        // handle error
      }
    }
```

This method does not establish an agent connection (see the code listings at the end of this article for examples of how this is done), nor

does it check the result of the start request (by reading the *pcfResponses* messages into an *MQCFH* data structure). It does, however, demonstrate two advantages of the Java classes: they are less verbose and are higher level than their C equivalents. Another benefit of the Java classes is that much of the tedium of using the C implementation (such as length setting and response parsing) is eliminated.

INSTALLING AND RUNNING PCF JAVA

Unfortunately, the downloadable IBM PCF support pack that I got was only available in **tar** format (this may have changed), which meant that I had to unpack it on a Linux machine. Perhaps because of this, the directory structure within the 'jar' that emerged was not present. I was, therefore, forced to 'unjar' the product and manually copy the components into their 'com\ibm\mq\pcf' hierarchy. Once this was done, I was able to place the PCF classes and the base MQ Java client classes into my *classpath* and run the sample code. Users of different versions of Unix may not have the same difficulty.

In order to accept PCF commands, a queue manager must satisfy the following criteria:

- The queue manager must be running.
- The default command queues must be in place (they are created in the default MQSeries Version 5 installation).
- A route to the queues must be available. In this case, client channels and the TCP channel listener (**runmqlsr**) are used.
- The command server must be started (using the command STRMQCSV *QueueManagerName*).

MQ CRAWLER

Based on our organization's requirements, I developed a basic framework of Java classes to help monitor the availability of MQSeries connections. MQSeries channels are invariably the weakest link in an MQSeries infrastructure, as they are subject to the relative vagaries of networking. I have, therefore, concentrated on automating the task of querying queue managers for the names of their sender channels and

their status. In this way, any channel that is not in a ‘running’ or ‘inactive’ state is treated as a potential problem.

In addition to this, the topology of a network of interconnected queue managers is often confusing. I have developed a technique, using basic MQ standards, for the auto discovery of remote queue managers and the clarification of the resource to which channels point that involves ‘crawling’ around from one queue manager to another.

The classes that implement this are presented at the end of this article. While they don’t include a front-end either to represent this data in a meaningful manner or to raise alerts, this is only the first part of an on-going project, and I intend to build on it in future articles.

MQ CRAWLER CLASSES

- *PCFHashtable*
  This is simply a utility wrapper class. It is created using a response message (for instance, the response from a ‘channel details’ enquiry), and then walks through a message and splits its components (such as MQCFINs or MQCFSTs) into entries in a hash table. Other classes are then able to query values without knowledge of parameter order. For example, if *iChannelDetails* is an instance of *PCFHashtable* (again created in response to a channel details query), the following code can be used to retrieve the channel type:

  ```
  int chltype = iChannelDetails.getIntValue(MQIACH_CHANNEL_TYPE);
  ```

- *ChannelDetailsPCF*
  This class is used to generate the PCF request that queries a queue manager for details of all its channels.

- *ChannelStatusPCF*
  This class is similar to *ChannelDetailsPCF*, with the exception that it generates a status query.

The remaining classes are wrappers for the main components of an MQSeries infrastructure:

- *MQInfrastructure*
  This class manages all queue managers. It is the initial class of the

MQ Crawler suite. Once the PCF classes are installed (as described earlier), and the MQ Crawler classes are added to the *classpath*, it may be executed using the command below (you have to substitute appropriate values for *QmgrName*, *port*, and *address*).

```
java com.dmitri.pcf.MQInfrastructure QmgrName port address
```

The *MQInfrastructure* class is created with a start point queue manager, and its job is to coordinate the discovery of all other queue managers. Future refinements to this class will include the ability to specify more than one start point.

- *QmgrManager*
  This class manages connections and interactions with a single queue manager. It opens the agent link to the queue manager and queries it for channel details using the *ChannelDetailsPCF* and *ChannelStatusPCF* classes. Once it has details of channels, it creates *ManagedChannel* objects for further interaction.

- *ManagedChannel*
  An instance of this class is created to handle each channel. It's a place-holder for future channel management, such as channel starts and resets. Current functionality is limited, but includes the ability to make a query about target queue managers. Target queue manager details are passed up the class hierarchy to *MQInfrastructure* as part of the auto discovery process.

FUTURE ENHANCEMENTS

MQ Crawler is a simple framework for implementing system management functions. To be really useful, however, it needs some or all of the following additional functionality:

- A graphical interface or a well-structured text interface. This would give a clearer representation of queue managers and their relationship to channels.

- The ability to add queue managers manually to the managed infrastructure.

- The addition of threads to improve performance. MQ Crawler

currently queries queue managers sequentially, thus spending much of its time waiting for replies.

- Facilities for generating alerts on channel failures etc. Such alerts could be visual, audible, implemented using SNMP traps, e-mail, etc.
- Facilities for correcting error conditions manually in response to alerts. Such facilities could include ones to handle:
  - Channel restarts
  - Channel resets
  - Channel commit/roll-back (for 'in-doubt' messages)
  - Transmission queue re-enabling.
- Automatic recovery from pre-defined error conditions.
- Greater tolerance of errors when connecting to and dealing with queue manager resources.

In addition to the system management features listed above, a full-featured MQSeries tool should also provide facilities for handling configuration. Most third-party system management products include the following configuration features:

- Channel/queue creation and modification.
- Creation and modification of other MQSeries objects, such as queue managers and processes.
- Facilities for copying resources from one queue manager to another.

CONCLUSION

It should be clear by now that PCF programming (when wrapped to a sufficient degree) is not an inordinately complex task. It should also be clear, however, that a full-featured MQSeries system management software can be large and time-consuming to write.

While an off-the-shelf package sounds inviting, one must be prepared to deal with a solution that is generic. My own experience of the MQSeries systems management market is not favourable. I saw nearly all the products that were available last year at the Transaction and Messaging Conference (September 1998) and concluded that practically none succeed in balancing complexity with scope of features.

So, if you have good development resources (Java in this case, although wrapping in C++ should also be productive) it could be viable to produce a simple system management suite internally. A full-featured system does not, at present, seem appropriate. This article is not the start of a Linux-like shareware solution, though I hope that it (and follow-up articles) will help other developers to pursue a solution along these lines.

SOURCE CODE FOR THE CLASSES

Note that some of the listings below include the continuation character, '➤', to indicate that one line of code maps to several lines of print.

PCFHASHTABLE.JAVA

```
package com.dmitri.pcf;

import com.ibm.mq.pcf.*;
import com.ibm.mq.*;
import java.util.*;
/**
*  @author Dmitri
*
*   This class extends Hashtable to provide PCF-specific processing.
*/
public class PCFHashtable extends Hashtable {

  private int iReasonCode;

/**
*   Constructor that sets up the data based on a message.
*/
  PCFHashtable(MQMessage message) {

    super();
    try {
```

```
      MQCFH cfh = new MQCFH(message);
      iReasonCode = cfh.reason;

      if (isValid()) {

        PCFParameter p;
        for (int i = 0; i < cfh.parameterCount; i++) {

          // Walk through the returned attributes
          p = PCFParameter.nextParameter (message);
          Integer key = new Integer(p.getParameter());
          put(key, p.getValue());
        }
      }
    } catch (Exception ex) {
      System.out.println(ex);
    }
  }

/**
*   Is the PCF request valid and, hence, may it be used?
*/
  public boolean isValid() {
    return iReasonCode == 0;
  }

/**
*   Returns the reason code for use by diagnostic processes.
*/
  public int getReasonCode(){
    return iReasonCode;
  }

/**
*   Returns the int value represented by the key supplied.
*/
  public int getIntValue(int key) {
    Integer i = (Integer) get(new Integer(key));
    return i.intValue();
  }
/**
*   Returns the int array represented by the key supplied.
*/
  public int [] getIntArray(int key) {
    return (int []) get(new Integer(key));
  }
/**
*   Returns the string value represented by the key supplied.
*/
  public String getStringValue(int key) {
```

```
    // mq returns padded strings.
    return ((String) get(new Integer(key))).trim();
  }
/**
*   Returns the string array represented by the key supplied.
*/
  public String [] getStringArray(int key) {
    String [] paddedStrings =  (String []) get(new Integer(key));
    String [] trimStrings = new String [paddedStrings.length];
    for (int i = 0; i < paddedStrings.length; i++) {
      trimStrings[i] = paddedStrings[i].trim();
    }
    return trimStrings;
  }
}
```

CHANNELDETAILSPCF.JAVA

```
package com.dmitri.pcf;
import com.ibm.mq.pcf.*;
import com.ibm.mq.*;

/**
*   @author Dmitri
*
*   This class is used to query queue managers for details of all
*   the channels they own.
*   This query is currently limited to sender channels.
*/
public class ChannelDetailsPCF implements CMQCFC {

  private PCFAgent iAgent;
  private PCFParameter []  iParameters;

/**
*   This constructor requires a valid queue manager agent. This
*   "channel details" request is currently only for sender channels
*   (channel type of MQCHT_SENDER), though this may easily be
*   changed to MQCHT_ALL.
*/
  public ChannelDetailsPCF(PCFAgent agent) {

    iAgent = agent;
    iParameters = new PCFParameter [] {
        new MQCFST (MQCACH_CHANNEL_NAME, "*"),
//      new MQCFIN (MQIACH_CHANNEL_TYPE, CMQXC.MQCHT_SENDER),
        new MQCFIN (MQIACH_CHANNEL_TYPE, CMQXC.MQCHT_ALL),
        new MQCFIL (MQIACF_CHANNEL_ATTRS, new int [] {MQIACF_ALL} ),
        };
```

```
  }

/**
*  This method returns an array of PCFHashtables. Each hashtable
*  represents a single channel. The hashtable holds all data
*  pertaining to the channel, and the data varies according to the
*  channel, so that, for example, the sender channel holds details
*  of target queue manager, while the receiver channel doesn't.
*/
  public PCFHashtable [] getChannels() {

    try {
      MQMessage [] pcfResponses = iAgent.send (MQCMD_INQUIRE_CHANNEL,
                                                    iParameters);
      PCFHashtable [] details = new PCFHashtable [pcfResponses.length];

      for (int i = 0; i < pcfResponses.length; i++) {
        details[i] = new PCFHashtable(pcfResponses[i]);
      }
      return details;

    } catch (Exception e) {
      e.printStackTrace();
    }
    return null;
  }
}
```

CHANNELSTATUSPCF.JAVA

```
package com.dmitri.pcf;
import com.ibm.mq.pcf.*;
import com.ibm.mq.*;
/**
*   @author Dmitri
*
*   This class is used to query channel status.
*/
public class ChannelStatusPCF implements CMQCFC {

  private PCFAgent iAgent;
  private PCFParameter []  iParameters;

/**
*   Constructor (requires a valid queue manager agent).
*/
  public ChannelStatusPCF(PCFAgent agent) {

    iAgent = agent;
```

```
    iParameters =   new PCFParameter [] {
            new MQCFST (MQCACH_CHANNEL_NAME, "*"),
            new MQCFIL (MQIACH_CHANNEL_INSTANCE_ATTRS, new int []
                                                    {MQIACF_ALL} ),
            };
  }

/**
*  This method returns an array of PCFHashtables. Each hashtable
*  contains parameters describing channel status.
*/
  public PCFHashtable [] getChannelStatus() {

    try {
      MQMessage [] pcfResponses = iAgent.send
                          (MQCMD_INQUIRE_CHANNEL_STATUS, iParameters);
      PCFHashtable [] details = new PCFHashtable [pcfResponses.length];

      for (int i = 0; i < pcfResponses.length; i++) {
        details[i] = new PCFHashtable(pcfResponses[i]);
      }
      return details;

    } catch (Exception e) {
      e.printStackTrace();
    }
  return null;
  }
}
```

MQINFRASTRUCTURE.JAVA

```
package com.dmitri.pcf;

import com.ibm.mq.*;
import java.util.*;
/**
*   @author Dmitri
*
*  This class is the top-level module for the MQ Infrastructure
*  monitor suite. It is a coordination and look-up point for other
*  classes, and makes use of a standard Singleton pattern to ensure
*  that it is not duplicated.
*/
public class MQInfrastructure {

  private static MQInfrastructure cInstance;
  private Hashtable iAllQueueManagers = new Hashtable();
```

```
/**
*  Singleton private constructor. This sets up the first queue manager
*  (specified by a command line argument), then calls the method
*  discoverAllQmgrs to crawl through the MQSeries infrastructure.
*/
  private MQInfrastructure(String startQmgrName, int startPort,
                                                  String startAddress) {
    QMgrManager mgr = new QMgrManager(startQmgrName, startPort,
                                                  startAddress);
    try {
      mgr.connect();
      mgr.queryChannels();

      iAllQueueManagers.put(mgr.getName(), mgr);
      mgr.discoverQmgrs(iAllQueueManagers);
      discoverAllQmgrs(iAllQueueManagers);

    } catch (MQException ex) {

      ex.printStackTrace();
    }
  }

/**
*   Provides access to the singleton instance.
*/
  public static MQInfrastructure instance() {
    return cInstance;
  }

/**
*   Provides access to the queue manager data.
*/
  public Hashtable getQueueManagers() {
    return iAllQueueManagers;
  }

/**
*  Iterate through the listed queue managers, connecting and querying
*  until no more are discovered. This could be described as a "crawler
*  method", as it uses a queue manager's references to other queue
*  managers for "auto discovery" (as long as conventions are followed
*  and queue managers are running).
*
*  I've opted for an interative rather than a recursive approach here
*  as it makes everything a bit more explicit.
*/
  private void discoverAllQmgrs(Hashtable qmgrStore) {

    int currentSize = qmgrStore.size();
```

```
    for (int i = 0; i != currentSize; ) {

      i = currentSize;

      for (Enumeration enum = qmgrStore.elements();
                                          enum.hasMoreElements(); ) {

        QMgrManager current = (QMgrManager) enum.nextElement();
        if (!current.isConnected()) {
        // This is an area that needs to be improved. If a queue
        // manager refuses a connection for whatever reason, this
        // code attempts to re-connect on each sweep. While this
        // may be desirable, it imposes a processing overhead.
          try {
            current.connect();
            current.queryChannels();
            current.discoverQmgrs(qmgrStore);
          } catch (MQException ex) {
            System.out.println(ex);
          }
        }
      }
      currentSize = qmgrStore.size();
    }
  }

/**
*  Bootstrap method. Requires a starting point (queue manager name
*  and the port and address on which it's running.
*/
  public static void main(String [] args) {

    if (args.length != 3) {
      remindOfUsage();
    }
    try {
      int port = Integer.parseInt( args[1] );
      cInstance = new MQInfrastructure( args[0], port, args[2]);
      printAllQueueManagerDetails();

    } catch (NumberFormatException nex) {
      remindOfUsage();
    }
  }

/**
*   A gentle reminder of how to use this program.
*/
  private static void remindOfUsage() {
```

```
    System.out.println("Usage:");
    System.out.println("java com.dmitri.pcf.MQInfrastructure
                                        QueueManagerName Port Address");
    System.out.println("Note: MQSeries will use port 1414 as default");

    System.exit(0);
  }

/**
*  This is a very simple way of rendering all the data that is
*  collected. It iterates through all queue manager data and dumps
*  it to system out.
*/
  private static void printAllQueueManagerDetails() {

    for (Enumeration enum = instance().getQueueManagers().elements();
                                          enum.hasMoreElements(); ) {

      QMgrManager current = (QMgrManager) enum.nextElement();
      System.out.println("---------------------------------------
                         ➤  -----------------");
      System.out.println("Queue manager : " + current.getName());
      System.out.println("");
      System.out.println("Has channels:");

      ManagedChannel [] chls = current.getChannels();
      for (int i = 0; i < chls.length; i++) {

        System.out.println( chls[i].getName() + " : " +
                                         chls[i].getChannelStatus() );
      }
    }
  }
}
```

QMGRMANAGER.JAVA

```
package com.dmitri.pcf;

import com.ibm.mq.pcf.*;
import com.ibm.mq.*;
import java.util.*;
/**
*  @author Dmitri
*
*  This class is used to control all access to managed MQSeries
*  queue managers. It controls access to resources.
*
*  This class relies both on queue managers being accessible via
```

```
*  client connections (TCP/IP) and the command server being running.
*/
public class QMgrManager implements CMQCFC {

  private String iHostname;
  private int iPort;
  private String iQmgrName;
  private String iClientChannel;
  private PCFAgent iAgent;
  private boolean iIsConnected = false;

  private ChannelDetailsPCF iDetailsPCF;
  private ChannelStatusPCF iStatusPCF;

  private Hashtable iManagedChannels = new Hashtable();

  public final static String CLIENT_CHANNEL = "SYSTEM.DEF.SVRCONN";

/**
*  The constructor (it assumes the default CLIENT_CHANNEL is used
*  to reach the destination qmgr).
*/
  public QMgrManager(String qmgrName, int port, String hostname) {
    this(qmgrName, hostname, port, CLIENT_CHANNEL);
  }

/**
*   The full constructor.
*/
  public QMgrManager(String qmgrName, String hostname, int port,
                                          String clientChannel) {

    iQmgrName = qmgrName;
    iHostname = hostname;
    iPort = port;
    iClientChannel = clientChannel;
  }

/**
*  Connects to the queue manager so that it can be queried. Also
*  prepares PCF queries.
*/
  public void connect() throws MQException {

    iAgent = new PCFAgent(iHostname, iPort, iClientChannel);
    iDetailsPCF = new ChannelDetailsPCF(iAgent);
    iStatusPCF = new ChannelStatusPCF(iAgent);
    iIsConnected = true;
  }
```

```
/**
*  Used to determine whether this is an active queue manager.
*/
  public boolean isConnected() {
    return iIsConnected;
  }

/**
*  Returns the queue manager's name.
*/
  public String getName() {
    return iQmgrName;
  }

/**
*  Query the queue manager for channel details.
*/
  public void queryChannels() {

    PCFHashtable [] details = iDetailsPCF.getChannels();
    PCFHashtable [] status = iStatusPCF.getChannelStatus();
    PCFHashtable current;
    String name;

    for (int i = 0; i < details.length; i++) {
      current = null;
      name = details[i].getStringValue(MQCACH_CHANNEL_NAME);

      for (int j = 0; j < status.length; j++) {
        // try to match status with details.
        if (name.equals(status[j].getStringValue(
                                              MQCACH_CHANNEL_NAME))) {
          current = status[j];
          break;
        }
      }
      ManagedChannel mchl = (ManagedChannel)
                               iManagedChannels.get(name);

      //instantiation is requred the first time channel is found.
      if (mchl == null) {
        mchl = new ManagedChannel(iAgent);
        iManagedChannels.put(name, mchl);
      }
      mchl.refreshChannel(details[i], current);

    }
  }

/**
```

```
*  This method may be called by a control object to query queue
*  managers for details of other queue managers, thereby discovering
*  ones not known to the control object. This is achieved by
*  delegating the call to the managed channels and querying them
*  for remote queue manager details. The hashtable returned is
*  keyed by queue manager name, and the item is a QMgrManager
*  object (not connected).
*/
  public void discoverQmgrs(Hashtable qmgrs) {

    ManagedChannel currentChl;

    for (Enumeration chls = iManagedChannels.elements();
                                        chls.hasMoreElements(); ) {

      currentChl = (ManagedChannel) chls.nextElement();
      currentChl.copyQmgrsInto(qmgrs);
    }
  }

/**
*   Returns an array of managed channels.
*/
  public ManagedChannel [] getChannels() {

    ManagedChannel [] channels = new ManagedChannel
                                          [iManagedChannels.size()];
    int i = 0;
    for (Enumeration chls = iManagedChannels.elements();
                                      chls.hasMoreElements(); i++) {

      channels[i] = (ManagedChannel) chls.nextElement();
    }
        return channels;

  }
}
```

MANAGEDCHANNEL.JAVA

```
package com.dmitri.pcf;

import com.ibm.mq.pcf.*;
import com.ibm.mq.*;
import java.util.*;
/**
*  @author Dmitri
*
*  This class is used to control access to channels. It deals with a
```

```
*  single channel and coordinates other facets of channel management.
*
*  This is the main class that would be extended to provide a more
*  complete system management solution. For example, from here it
*  would be possible to add a channel renderer to represent a channel
*  visually. It would also be appropriate to tag on classes to alter
*  channels (eg by starting or resetting them).
*/
public class ManagedChannel implements CMQCFC, CMQXC {

  private PCFHashtable iChannelDetails;
  private PCFHashtable iChannelStatus;
  private String iName;
  private String iXmitQName;
  private PCFAgent iAgent;

  private QMgrManager iTargetQMgr;

  public static int DEFAULT_PORT = 1414;

  // lookup point to translate channel status into English.
  public static String [] CHANNEL_STATUS = {
    "Inactive",
    "Binding",
    "Starting",
    "Running",
    "Stopping",
    "Retrying",
    "Stopped",
    "Requesting",
    "Paused",
    "",
    "",
    "",
    "",
    "Initializing",
  };

/**
*  Simple constructor that saves a reference to the queue manager's
*  agent.
*/
  public ManagedChannel(PCFAgent agent) {
    super();
    iAgent = agent;
  }

/**
*  This method is called by the QMgrManager class. This is done
*  because calls to enquire channel details and status may return
```

```
*  multiple channels.
*/
  public void refreshChannel(PCFHashtable details,
                                                PCFHashtable status) {

    iChannelDetails = details;
    iChannelStatus = status; // often null.

    iName = iChannelDetails.getStringValue(MQCACH_CHANNEL_NAME);
    refreshXmitQ();
    refreshTargetQmgr();
  }

/**
*   Return the name of the managed channel.
*/
  public String getName() {
    return iName;
  }

/**
*   Return the channel's status (string format).
*/
  public String getChannelStatus() {
    return CHANNEL_STATUS[ getChannelStatusInt() ];
  }

/**
*   Return the channel's status (int format).
*/
  public int getChannelStatusInt() {

    try {
      return iChannelStatus.getIntValue(MQIACH_CHANNEL_STATUS);

    } catch (NullPointerException channelIsInactive) {
      return 0;
    }

  }

/**
*  A possible future extension could be to get the depth of the XmitQ.
*  An XmitQ with messages on is a good sign of problems in a system
*  that requires channels to be running at all times.
*/
  private void refreshXmitQ() {

    try {
      iXmitQName = iChannelDetails.getStringValue(MQCACH_XMIT_Q_NAME);
```

```
      } catch (NullPointerException noXmitq) {
    }
  }

/**
*  This is used to discover other queue managers on a network. It
*  does this by querying sender/server channels for details of
*  partner queue managers, and makes a number of assumptions:
*
*  1  That the connection is based on TCP/IP (the whole Java MQ
*     interface relies on IP).
*  2  That the convention is followed whereby XMitQs have the same
*     name as queue managers (MQSeries management becomes difficult
*     if this convention is not followed).
*  3  That remote queue managers are accessible via the default
*     client connection channel.
*/
  private void refreshTargetQmgr() {

    if (iXmitQName == null) return;

    // If already set up...
    if (iTargetQMgr != null &&
                          TargetQMgr.getName().equals(iXmitQName)) {
      return;
    }
    int chltype = iChannelDetails.getIntValue(MQIACH_CHANNEL_TYPE);
    if (chltype != MQCHT_SENDER &&
                chltype != MQCHT_SERVER) { // we are unable to query
                                           // for connection name.
      return;
    }
    String conname =
              iChannelDetails.getStringValue(MQCACH_CONNECTION_NAME);
    int port = DEFAULT_PORT;

    if (conname.indexOf('(') > 0) {   // port number is embedded
      port = Integer.parseInt(   conname.substring(
                  conname.indexOf('(') + 1, conname.indexOf(')') ) );
      conname = conname.substring( 0, conname.indexOf('(') );
    }

    iTargetQMgr = new QMgrManager( iXmitQName, port, conname );

  }

/**
*  A method of accumulating all data about referenced queue
*  managers in a common place.
*/
```

```
  public void copyQmgrsInto(Hashtable qmgrs) {

    if (iTargetQMgr == null ||
      iTargetQMgr.getName().equals("") ||
      qmgrs.containsKey(iTargetQMgr.getName())) {
      return;
    }
    qmgrs.put(iTargetQMgr.getName(), iTargetQMgr);
  }
}
```

STARTCHANNELPCF.JAVA

```
package com.dmitri.pcf;
import com.ibm.mq.pcf.*;
import com.ibm.mq.*;

/**
*   @author Dmitri
*
*  This class is used to start channels. It is not linked to other
*  classes and is included only as an example.
*/
public class StartChannelPCF implements CMQCFC {

  private PCFAgent iAgent;
  private PCFParameter []  iParameters;

/**
*  The constructor, which requires a valid queue manager agent.
*/
  public StartChannelPCF(PCFAgent agent) {

    iAgent = agent;
  }

/**
*  This method starts the channel specified.
*/
  public void startChannel(String channelName) {

    iParameters =    new PCFParameter [] {
            new MQCFST (MQCACH_CHANNEL_NAME, channelName),
            };

    try {
      MQMessage [] pcfResponses = iAgent.send (MQCMD_START_CHANNEL,
                                                    iParameters);
```

```
      // Assume only the one response.
      PCFHashtable response = new PCFHashtable(pcfResponses[0]);

      if (response.isValid()) {
        System.out.println("Channel " + channelName + " Started");

      } else {
        System.out.println( "Channel " + channelName +
                            " failed to start, reason code : " +
                            response.getReasonCode() );
      }

    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```


Copying object definitions from QMGR to QMGR

Part of MQSeries for MVS/ESA is a set of operations and control panels to construct and run commands for defining, displaying, altering, and deleting MQSeries objects under ISPF. But I was missing a function for copying object definitions from one queue manager to another (for instance, from TEST to PROD). To plug this gap I wrote MQSMAKE EXEC (it comprises one REXX EXEC and two ISPF panels), which uses the COMMAND function of the CSQUTIL utility program to produce a list of object DEFINE statements and pass them to a target subsystem. As the CSQUTIL load module is called as an external routine, the appropriate MQSeries libraries (thlqual.SCSQAUTH and thlqual.SCSQANLE) must be included in the TSO STEPLIB concatenation. Figure 1 overleaf shows the processes involved.

Initially a selection panel (MQSMAKE) asks for the source and target queue manager and also for the objects for which DEFINE statements should be generated. After receiving the arguments, temporary data

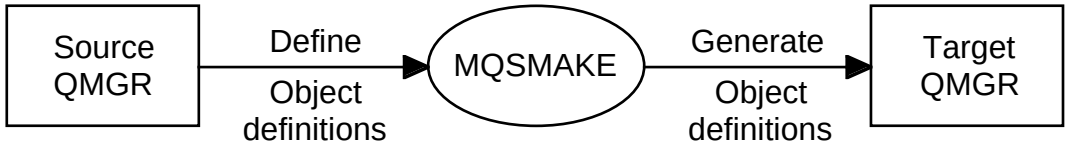

Figure 1: Steps performed by MQSMAKE

sets for use by the utility program are allocated and CSQUTIL is called for the first time to generate DEFINE statements from the specified DISPLAY object commands. If any statements are generated, the MQSMAKE2 panel prompts you to edit the MAKEDEF output and execute the define step. Otherwise, the queue manager utility SYSPRINT is displayed and the dialogue function returns to the MQSMAKE panel. During editing, the DEFINE statements generated can be modified before they are passed to the target queue manager via a second call to CSQUTIL. If the return code from the call command signals successful processing, SYSPRINT is displayed for verifying the DEFINE object execution. Returning (PF3-End) to the selection screen (MQSMAKE) deletes all temporary data sets.

Note the use of the continuation character ('➤') in the code below to indicate that one line of code maps to more than one line of print. This character is not present in the code downloadable from Xephon's Web site (*www.xephon.com*).

REXX SOURCE CODE

```
EXEC MQSMAKE
/* REXX */
/*====================================================================-/
/- Author  : R.Kleebaur                                               -/
/- Date    : 10.01.1999                                               -/
/- Function: Copy object definitions from QMGR to QMGR via CSQUTIL  -/
/-           utility. The user is prompted before the selected      -/
/-           object(s) are defined in target QMGR.                  -/
/-====================================================================-*/
fqmgr    = ''                                /* init panel fields   */
tqmgr    = ''
queue    = ''
namelist = ''
process  = ''
channel  = ''
```

```
msg      = ''
lib      = 'THQ.SCSQAUTH'                    /* CSQUTIL library     */
do forever
    ADDRESS ISPEXEC 'DISPLAY PANEL (MQSMAKE)'
    if rc = 8 then
       leave                                 /* pf3 - end           */
    else
     do
       alloc_rc  = 0
       x = outtrap(var.,'*')
       call alloc                  /* allocate temporary data sets  */
       if alloc_rc = 0 then
          do
            call genstep           /* generate object definitions   */
            if  genstep_rc = 0 then
                do
                  call defprompt   /* should defstep be performed ? */
                end
            else
                do
                  call dealloc     /* delete temorary data sets     */
                  msg = 'Generate step unsuccessful |'
                end
          end
     end
end
exit
/*--------------------------------------------------------------------*/
/- subroutine: allocate temporary data sets for utility execution   -/
/-             dsntemp1 = SYSIN                                     -/
/-             dsntemp2 = SYSPRINT                                  -/
/-             dsntemp3 = CMDINPUT  input                           -/
/-             dsntemp4 = MAKEDEF   output                          -/
/--------------------------------------------------------------------*/
alloc:
temp1 = 'SYSIN'
temp2 = 'SYSPRINT'
temp3 = 'CMDINPUT'
temp4 = 'MAKEDEF'
dsntemp1 = USERID() || '.MQSMAKE' || '.SYSIN'
dsntemp2 = USERID() || '.MQSMAKE' || '.SYSPRINT'
dsntemp3 = USERID() || '.MQSMAKE' || '.CMDINPUT'
dsntemp4 = USERID() || '.MQSMAKE' || '.MAKEDEF'

/* allocate temp dataset 1 */
ADDRESS TSO "ALLOC FI("temp1") DA('"dsntemp1"') OLD REUSE"
if rc = 0 then
   ADDRESS TSO "EXECIO 0 DISKW "temp1" (OPEN FINIS"
else
   ADDRESS TSO "ALLOC FI("temp1") DA('"dsntemp1"'),
```

```
   NEW CAT REUSE UNIT(SYSTS),
   LRECL(80) BLKSIZE(27920) RECFM(F B) SPACE(1,1) TRACKS"
   if rc /= 0 then
      do
        msg = 'Temporary dataset 1 cannot be allocated |'
        alloc_rc = 1
        return
      end

/* allocate temp dataset 2 */
ADDRESS TSO "ALLOC FI("temp2") DA('"dsntemp2"') OLD REUSE"
if rc = 0 then
   ADDRESS TSO "EXECIO 0 DISKW "temp2" (OPEN FINIS"
else
   ADDRESS TSO "ALLOC FI("temp2") DA('"dsntemp2"'),
   NEW CAT REUSE UNIT(SYSTS),
   LRECL(133) BLKSIZE(27930) RECFM(F B) SPACE(1,1) TRACKS"
   if rc /= 0 then
      do
        msg = 'Temporary dataset 2 cannot be allocated |'
        alloc_rc = 1
        return
      end

/* allocate temp dataset 3 */
ADDRESS TSO "ALLOC FI("temp3") DA('"dsntemp3"') OLD REUSE"
if rc = 0 then
   ADDRESS TSO "EXECIO 0 DISKW "temp3" (OPEN FINIS"
else
   ADDRESS TSO "ALLOC FI("temp3") DA('"dsntemp3"'),
   NEW CAT REUSE UNIT(SYSTS),
   LRECL(80) BLKSIZE(27920) RECFM(F B) SPACE(1,1) TRACKS"
   if rc /= 0 then
      do
        msg = 'Temporary dataset 3 cannot be allocated |'
        alloc_rc = 1
        return
      end

/* allocate temp dataset 4 */
ADDRESS TSO "ALLOC FI("temp4") DA('"dsntemp4"') OLD REUSE"
if rc = 0 then
   ADDRESS TSO "EXECIO 0 DISKW "temp4" (OPEN FINIS"
else
   ADDRESS TSO "ALLOC FI("temp4") DA('"dsntemp4"'),
   NEW CAT REUSE UNIT(SYSTS),
   LRECL(80) BLKSIZE(27920) RECFM(F B) SPACE(1,1) TRACKS"
   if rc /= 0 then
      do
        msg = 'Temporary dataset 4 cannot be allocated |'
```

```
        alloc_rc = 1
        return
      end
return

/*-------------------------------------------------------------------*/
/- subroutine: reuse temp data sets for define step                  -/
/-             dsntemp1 = SYSIN                                      -/
/-             dsntemp2 = SYSPRINT                                   -/
/-------------------------------------------------------------------*/
alloc_reuse:
ADDRESS TSO "ALLOC FI("temp1") DA('"dsntemp1"') OLD REUSE"
ADDRESS TSO "EXECIO 0 DISKW "temp1" (OPEN FINIS"
ADDRESS TSO "ALLOC FI("temp2") DA('"dsntemp2"') OLD REUSE"
ADDRESS TSO "EXECIO 0 DISKW "temp2" (OPEN FINIS"
return
/*-------------------------------------------------------------------*/
/- subroutine: dealloc and delete temp data sets                     -/
/-------------------------------------------------------------------*/
dealloc:
ADDRESS TSO "FREE  FI("temp1")"
ADDRESS TSO "FREE  FI("temp2")"
ADDRESS TSO "FREE  FI("temp3")"
ADDRESS TSO "FREE  FI("temp4")"
ADDRESS TSO "DELETE ('"dsntemp1"')"
ADDRESS TSO "DELETE ('"dsntemp2"')"
ADDRESS TSO "DELETE ('"dsntemp3"')"
ADDRESS TSO "DELETE ('"dsntemp4"')"
return
/*-------------------------------------------------------------------*/
/- subroutine: generate MQS object definitions                       -/
/-------------------------------------------------------------------*/
genstep:
queue 'COMMAND DDNAME(' || temp3 || ') MAKEDEF(' || temp4 || ')'
queue ''                                    /* enter null line   */
ADDRESS TSO "EXECIO * DISKW "temp1          /* write SYSIN       */
ADDRESS TSO "EXECIO 0 DISKW "temp1" (FINIS"    /* close SYSIN       */
if queue    ¬= '' then
   queue  'DISPLAY QUEUE(' || queue || ') ALL'
if namelist ¬= '' then
   queue  'DISPLAY NAMELIST(' || namelist || ') ALL'
if process  ¬= '' then
   queue  'DISPLAY PROCESS(' || process || ') ALL'
if channel  ¬= '' then
   queue  'DISPLAY CHANNEL(' || channel || ') ALL'
queue ''                                    /* enter null line   */
ADDRESS TSO "EXECIO * DISKW "temp3          /* write CMDINPUT    */
ADDRESS TSO "EXECIO 0 DISKW "temp3" (FINIS"    /* close CMDINPUT    */
ADDRESS TSO "call '" || lib || "(CSQUTIL)' '" || fqmgr || "'"
genstep_rc = rc                             /* save return code  */
```

```
return
/*-------------------------------------------------------------------*/
/- subroutine: define objects in target qmgr                        -/
/-             1. analyse genstep SYSPRINT                         -/
/-             2. display defstep execution prompt panel           -/
/-             3. execute defstep                                  -/
/*-------------------------------------------------------------------*/
defprompt:
call genstep_analyse
if cmdno = 0 then
   do
     ADDRESS ISPEXEC "VIEW DATASET('"dsntemp2"')" /* displ SYSPRINT */
     call dealloc                              /* del temp datasets */
     msg = 'No DEFINE commands generated |'
     return
   end
defmsg  = cmdno 'DEFINE command(s) generated |'
makedef = 'Y'
defstep = 'Y'
ADDRESS ISPEXEC "ADDPOP"
ADDRESS ISPEXEC "DISPLAY PANEL(MQSMAKE2)"        /* prompt exec'n   */
ADDRESS ISPEXEC "REMPOP"
if makedef = 'Y' then                            /* edit MAKEDEF?   */
   ADDRESS ISPEXEC "EDIT DATASET('"dsntemp4"')"
if defstep = 'Y' then                            /* execute define? */
   do
     call alloc_reuse                            /* reuse data sets */
     call defstep                                /* run utility     */
     if  defstep_rc = 0 then                     /* define executed */
         do                                      /* displ SYSPRINT  */
           ADDRESS ISPEXEC "VIEW DATASET('"dsntemp2"')"
           call dealloc                          /* del temp datasets */
           msg = 'Define step executed |'
         end
     else                                      /* define unsuccessful */
         do
           call dealloc                          /* del temp datasets */
           msg = 'Define step unsuccessful |'
         end
   end
else                                             /* define skipped    */
   do
    call dealloc                                 /* del temp datasets */
    msg = 'Define step skipped |'
   end
return
/*-------------------------------------------------------------------*/
/- subroutine: extract CSQU059I message from genstep SYSPRINT       -/
/*-------------------------------------------------------------------*/
genstep_analyse:
```

```
ADDRESS TSO "EXECIO * DISKR "temp2" (STEM recin. FINIS"
cmdno = 0                                  /* no of define cmds made */
do i = 1 to recin.0
   if pos('CSQU059I',recin.i) ¬= 0 then
      do
         parse var recin.i 'CSQU059I' . cmdno .
         leave
      end
end
return
/*--------------------------------------------------------------------/
/- subroutine: define MQS objects in target QMGR                     -/
/--------------------------------------------------------------------*/
defstep:
queue 'COMMAND DDNAME(' || temp4 || ')'
queue ''                                        /* enter null line   */
ADDRESS TSO "EXECIO * DISKW "temp1              /* write SYSIN       */
ADDRESS TSO "EXECIO 0 DISKW "temp1" (FINIS"     /* close SYSIN       */
ADDRESS TSO "call '" || lib || "(CSQUTIL)' '" || tqmgr || "'"
defstep_rc = rc                                 /* save return code  */
return
```

ISPF PANELS

```
PANEL MQSMAKE
)Body
%               COPY OBJECT DEFINITION(S)                    User
➤  -+&ZUSER
%                                                            Date
➤  -+&date
% COMMAND ===> _ZCMD                                        +%Time
➤  -+&ZTIME
%-------------------------------------------------------------------
➤  ------------
+
+
+        %From ........:+ _FQMGR +     %To ........:+ _TQMGR +
+
+        %For object(s):
+        %Queue .......:+ _QUEUE
➤         +
+        %Namelist ....:+ _NAMELIST
➤         +
+        %Process .....:+ _PROCESS
➤         +
+        %Channel .....:+ _CHANNEL            +
+
+        %Note: Generic selection possible. There will be a prompt
➤  before
```

```
+        %      the define step.
+
+
+
+         &MSG
%-------------------------------------------------------------------
➤  ------------
%
%
)INIT
   .CURSOR = FQMGR
   &DATE    = '&ZDAY..&ZMONTH..&ZSTDYEAR'
)PROC
   VER (&fqmgr,nb)
   VER (&tqmgr,nb)
   IF  (&fqmgr = &tqmgr)
       .CURSOR=TQMGR
       .MSG=MQM000
   IF  (&queue = '' & &namelist = '' & &process = '' & &channel = '')
       .CURSOR=QUEUE
       .MSG=MQM001
)END

PANEL MQSMAKE2
)Body  window(65,10)
%COMMAND ===>_ZCMD
%-------------------------------------------------------------------
+             &defmsg
+
+            %Edit MAKEDEF output:+ _makedef +
+            %Perform define step:+ _defstep +
%-------------------------------------------------------------------
)INIT
 .CURSOR = ZCMD
)REINIT
 REFRESH(*)
)PROC
   VER (&makedef,list,Y,N)
   VER (&defstep,list,Y,N)
)END

MESSAGE MEMBER MQM00
MQM000  'INVALID SELECTION' .TYPE=WARNING
MQM001  'NO OBJECT SELECTED' .TYPE=WARNING
```

Raimund Kleebaur
MQSeries Programmer
Hugo Boss AG (Germany)

MQSeries stress testing

This months's instalment concludes this article on MQSeries stress testing using Java (the first part appeared in last month's *MQ Update*).

MQCONSTANTS.JAVA (CONTINUED)

```
JPanel bottom = new JPanel();
    dlgContents.add(bottom, BorderLayout.SOUTH);
    bottom.setLayout(new FlowLayout());
    JButton btnOk = new JButton("Ok");
    bottom.add(btnOk);
    btnOk.addActionListener( new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        iMessagedlg.setVisible(false);
      }
    });
    iMessagedlg.pack();
    iMessagedlg.setVisible(true);
  }

/**
*   Return the singleton's instance
*/
  public static synchronized MQConstants getInstance() {

    if (cInstance == null) {

      try {
        cInstance = new MQConstants(); // Swing styles must be
                                       // re-instantiated.
        ObjectInputStream ois = new ObjectInputStream( new
                                  FileInputStream(PERSISTENCY_NAME) );
        cInstance = (MQConstants) ois.readObject();
        cInstance.addListeners();
        ois.close();
      } catch (Exception e) {
        cInstance = new MQConstants();
      }

      cInstance.setVisible(true);
      cInstance.repaint();
    }
    return cInstance;
  }

/**
```

```
*   Store a reference to any classes interested in the completion
*   of the gathering of constants
*/
  public void addConstantsListener(ConstantsListener lis) {
    iListeners.addElement(lis);
  }
/**
*   Notify listeners that all constants have now been gathered
*/
  public void notifyConstantsListeners() {
    for (Enumeration enum = iListeners.elements();
                                    enum.hasMoreElements(); ) {
      ((ConstantsListener) enum.nextElement()).constantsAvailable();
    }
  }
}
```

CONSTANTSLISTENER.JAVA

```
package com.dmitri.mqstress;
/**
*   @author Dmitri
*  Simple interface to notify that constants are now ready for use
*/
public interface ConstantsListener {

  public void constantsAvailable();
}
```

MQSTRESSTESTER.JAVA

```
package com.dmitri.mqstress;
import java.awt.event.*;
import com.sun.java.swing.*;
import java.awt.*;
/**
*   @author Dmitri
*
*   This is the top-level class of the MQStressTester package. It
*   has overall responsibility for other classes.
*
*  Its primary actions are:
*
*  1. Instantiate the constants window and wait for it to finish
*     its information gathering and processing.
*  2. Instantiate an appropriate number of MQStressThreads.
*  3. Present the user with controls for starting and stopping
*     the test.
*  4. Control the starting and stopping of the stress test.
```

```
*/
public class MQStressTester implements ConstantsListener {

  private MQConstants iConstants = MQConstants.getInstance();

  private JFrame iFrmResults = new JFrame("MQStress Results");
  private JButton iBtnStart = new JButton("Start");
  private JButton iBtnStop = new JButton("Stop");
  private JButton iBtnExit = new JButton("Exit");

  private static long cRepaintInterval = 1000l;
  private MQStressThread [] iThreads;    // An array of test threads

/**
*   A simple constructor to register an interest in the completion
*   of MQConstants. As we are dependent on constants being gathered,
*   it is necessary to wait until they are all in place.
*/
  public MQStressTester() {
    super();
    iConstants.addConstantsListener(this);
  }

/**
*   This method starts things off when the MQConstants class has
*   finished gathering data.
*/
  public void constantsAvailable() {
    begin();
  }

/**
*   This method initializes the window for displaying the results
*   of the stress test while in progress.
*/
  private void begin() {

    int numberOfThreads = iConstants.getNumberOfInstances();
    iThreads = new MQStressThread[numberOfThreads];

    JPanel pnlContents = new JPanel();
    JPanel pnlMain = new JPanel();
    iFrmResults.getContentPane().add(pnlContents);
    pnlContents.setBorder(BorderFactory.createEmptyBorder(
                                                    10, 10, 10, 10));
    pnlContents.setLayout(new BorderLayout());
    pnlContents.add(pnlMain);
    pnlMain.setLayout(new FlowLayout());

    for (int i = 0; i < numberOfThreads; i++) {
```

```
      iThreads[i] = new MQStressThread();
      pnlMain.add(iThreads[i].getRenderer());
    }

    JPanel pnlButtons = new JPanel();
    pnlButtons.setLayout(new FlowLayout());
    pnlContents.add(pnlButtons, BorderLayout.SOUTH);
    pnlButtons.add(iBtnStart);
    pnlButtons.add(iBtnStop);
    pnlButtons.add(iBtnExit);

    addListeners();
    iFrmResults.pack();
    iFrmResults.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    iFrmResults.setVisible(true);
  }

/**
*   This method adds inner class listeners to all buttons.
*/
  private void addListeners() {

    iBtnStart.addActionListener( new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        for (int i = 0; i < iThreads.length; i++) {
          new Thread( iThreads[i] ).start();
        }
      }
    });

    iBtnStop.addActionListener( new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        for (int i = 0; i < iThreads.length; i++) {
          iThreads[i].stopTest();
        }
      }
    });

    iBtnExit.addActionListener( new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        for (int i = 0; i < iThreads.length; i++) {
          iThreads[i].exit();
          iFrmResults.setVisible(false);
          System.exit(0);
        }
      }
    });
  }

/**
*   This is a standard main method to instantiate this class.
```

```
*/
  public static void main(String[] args) {
    new MQStressTester();
  }
}
```

MQSERVICE.JAVA

```
package com.dmitri.mqstress;
import com.ibm.mq.*;
/**
*   @author Dmitri
*
*   This is a simple class that interfaces to MQSeries, putting
*   and getting simple messages. It doesn't act on messages. This
*   class is not capable of recovering from MQSeries errors (such
*   as connection lost).
*/
public class MQService {

  private MQConstants iConstants = MQConstants.getInstance();
  private MQQueue iRequestQueue = null;
  private MQQueue iReplyQueue = null;
  private MQGetMessageOptions iGmo = new MQGetMessageOptions();
  private MQPutMessageOptions iPmo = new MQPutMessageOptions();
  private MQQueueManager iQmgr;

/**
*  Initializes a new service to MQSeries
*/
  public MQService() {
    System.runFinalizersOnExit(true);
  }

/**
 * This method sets up the default message parameters. Some of these
 * are hard-coded, others are obtained from the constants class.
 */
  private MQMessage applyDefaults(String msg) {

    try {
      MQMessage mqmess = new MQMessage();
      mqmess.replyToQueueName = iReplyQueue.name;
      mqmess.format = MQC.MQFMT_STRING;
      mqmess.writeString(msg);

      return mqmess;

    } catch (java.io.IOException e) {
      e.printStackTrace();
```

```
    }
    return null;
  }

/**
 * This method initializes the MQ component of the service.
 * This includes connection to the queue manager as well as the
 * opening queues.
 *
 * Note: for future expansion, it may be a good idea to measure
 * connection times etc, as this takes an appreciable time in a
 * stressed system.
 */
  public void mqStartup() {

    iGmo.options = MQC.MQGMO_WAIT | MQC.MQGMO_CONVERT |
                   MQC.MQGMO_NO_SYNCPOINT | MQC.MQGMO_FAIL_IF_QUIESCING;
    iPmo.options = MQC.MQPMO_NO_SYNCPOINT | MQC.MQPMO_FAIL_IF_QUIESCING
              | MQC.MQPMO_SET_IDENTITY_CONTEXT;

    MQEnvironment.hostname = iConstants.getHostName();
    MQEnvironment.channel  = iConstants.getChannelName();

//  Connect to the specified queue manager
    try {
      iQmgr = new MQQueueManager( iConstants.getQueueManagerName() );

      int openOptions = MQC.MQOO_OUTPUT | MQC.MQOO_FAIL_IF_QUIESCING
                | MQC.MQOO_SET_IDENTITY_CONTEXT;
// Open request queue
      iRequestQueue = iQmgr.accessQueue(
                                     iConstants.getRequestQueueName(),
                                      openOptions, null, null, null);

      openOptions = MQC.MQOO_INPUT_SHARED | MQC.MQOO_FAIL_IF_QUIESCING;

// Open reply queue
      iReplyQueue = iQmgr.accessQueue( iConstants.getReplyQueueName(),
                                       openOptions, null, null, null);
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }

/**
 * This method places a message on a queue and waits for a reply
 * on the reply queue. It returns the time taken or -1 if it fails.
 */
  public long sendAndReceive(String ioutMessage, long timeOut) {

    iGmo.waitInterval = (int) timeOut;
```

```
    MQMessage outMess = applyDefaults(ioutMessage);
    long timeStart = System.currentTimeMillis();
    try {
      iRequestQueue.put(outMess, iPmo);
      iReplyQueue.get( outMess, iGmo,
                              iConstants.getMaximumMessageLength() );

      return System.currentTimeMillis() - timeStart;

    } catch (Exception e) {
      System.out.println("Put/Get error : " + e);
    }
    return -1;
  }

/**
*   Close down the mq queues and connection
*/
  public void finalize() {

    try {
      iReplyQueue.close();
      iRequestQueue.close();
      iQmgr.disconnect();
    } catch (MQException tooLateToWorry) {}
  }
}
```

MQSTRESSTHREAD.JAVA

```
package com.dmitri.mqstress;
import java.awt.*;
import com.sun.java.swing.*;

/**
*   @author Dmitri
*
*  This class is the actual control object for the MQStress test
*  package. It calls the MQService class and reports results back
*  to the main class by supplying a component to hold results.
*
*   In this implementation, a simple JLabel is supplied to show
*   test results as they happen. This gives an on-line view that
*   immediately highlights problem areas. This implementation does
*   not store results of any kind. It could be appropriate to extend
*   this class to do just that, enabling extensive analysis of
*   results at a later stage.
*/
public class MQStressThread implements Runnable {

  private JLabel iRenderer = new JLabel("Inactive");
```

```
  private MQService iService = new MQService();

  private boolean iStopRequested = false;  // Flag to determine
                                           // whether the control module
                                           // has requested a stop.

  private MQConstants iConstants = MQConstants.getInstance();

  private long [] iAcceptable;
  private long [] iFailure;
  private long [] iPace;
  private String [] iMessage;

  private Color iClrAcceptable = new Color(0, 70, 0); // Dark green
  private Color iClrWarning = new Color(170, 170, 0);  // Yellowish
  private Color iClrFail = Color.red;
  private Color iClrDefault = Color.black;

/**
*    Constructor. Initializes local copies of runtime parameters to
*    avoid delays when running.
*/
  public MQStressThread() {

    iRenderer.setPreferredSize(new Dimension(60,20));
    iRenderer.setForeground(iClrDefault);
    int messNo = iConstants.getNumberOfMesages();
    iAcceptable = new long[messNo];
    iFailure = new long[messNo];
    iPace = new long[messNo];
    iMessage = new String[messNo];

    for (int i = 0; i < messNo; i++) {
      iAcceptable[i] = iConstants.getAcceptableTime(i);
      iFailure[i] = iConstants.getFailureTime(i);
      iPace[i] = iConstants.getPaceTime(i);
      iMessage[i] = iConstants.getMessage(i);
    }
    iService.mqStartup();
  }

/**
*    This method returns the 'renderer' that is used to display the
*    results of the stress test as they happen.
*/
  public Component getRenderer() {
    return iRenderer;
  }

/**
*    Starts the stress test. Records response times and paces
```

```
*    according to pre-set parameters.
*/
  public void run() {

    iStopRequested = false;
    int numberOfRepetitions = iConstants.getNumberOfRepetitions();
    int numberOfMessages = iConstants.getNumberOfMesages();

    long startTime = 0;
    long response = 0;
    long sleepTime = 0;

    for (int i = 0; i < numberOfRepetitions && !iStopRequested; i++) {

      for (int j = 0; j < numberOfMessages && !iStopRequested; j++) {

        startTime = System.currentTimeMillis();
        response = iService.sendAndReceive(iMessage[j], iPace[j]);
                                  // Maximum time allowed is pace time.

        // Could log to disk here in future extensions.

        if (response < 0) {       // Complete failure
          iRenderer.setForeground(iClrFail);
          iRenderer.setText("Failure");

        } else if (response < iAcceptable[j]) {
          iRenderer.setForeground(iClrAcceptable);
          iRenderer.setText(Long.toString(response));

        } else if (response < iFailure[j]) {
          iRenderer.setForeground(iClrWarning);
          iRenderer.setText(Long.toString(response));

        } else if (response > iFailure[j]) {
          iRenderer.setForeground(iClrFail);
          iRenderer.setText(Long.toString(response));

        }
        sleepTime = iPace[j] - ( System.currentTimeMillis()
                                                  - startTime );
        try {
           Thread.sleep(sleepTime);
        } catch (Exception ignored) {}  // Could be an interrupt or
                                        // negative sleep
      }
    }
  }
/**
*    Stops the stress test
*/
```

```
  public void stopTest() {
    iStopRequested = true;
    iRenderer.setText("Stop");
    iRenderer.setForeground(iClrDefault);
  }
/**
*    Prepares for exiting
*/
  public void exit() {
    stopTest();
  }
}
```


Recovering damaged or lost circular logs on Unix

MQSeries builds its own log files in */var/mqm/log/QMGRNAME/active*. If the logs are defined as 'circular' you need never worry about them – MQ will maintain them itself. To find out how your logs are defined, check the *LogType* parameter in the */var/mqm/qmgrs/QMGRNAME/qm.ini* file (note that 'CIRCULAR' is the default).

Here's an excerpt from *qm.ini*:

```
Log:
     LogPrimaryFiles=3
     LogSecondaryFiles=2
     LogFilePages=1024
     LogType=CIRCULAR
     LogBufferPages=17
     LogPath=/var/mqm/log/QMGRNAME/
```

This is the log file structure:

Path: */var/mqm/log/QMGRNAME*:

```
drwxrwx---   3 mqm   mqm       512   May 26  1998  .
drwxrwxr-x   5 mqm   mqm       512   Jul 16 14:41  ..
drwxrwx---   2 mqm   mqm       512   May 26  1998  active
-rw-rw----   1 mqm   mqm      7580   Jul 16 14:25  amqhlctl.lfh
```

Path: */var/mqm/log/QMGRNAME/active*:


```
drwxrwx---   2 mqm   mqm       512   May 26  1998  .
drwxrwx---   3 mqm   mqm       512   May 26  1998  ..
-rw-rw----   1 mqm   mqm   4202496   Jul 16 14:25  S0000000.LOG
-rw-rw----   1 mqm   mqm   4202496   Jul 11 12:01  S0000001.LOG
-rw-rw----   1 mqm   mqm   4202496   Jul 14 09:23  S0000002.LOG
```

Occasionally, the logs are damaged or (if outside forces enforce a rigorous log archiving routine) they may be removed. Whatever the cause, the result is the same: MQ dies.

A quick way to make MQ operative again is to create a dummy queue manager and steal its logs:

1 Stop MQSeries, if it is not already down:

```
endmqm -i QMGRNAME
```

2 Create a dummy queue manager:

```
crtmqm DUMQMGR
```

3 Back up (rename) existing log files:

```
mv old.fil   /tmp/backup.name
```

4 Copy the log files and the log control record from the newly created queue manager (note the use of the continuation character, '➤', to indicate that one command line maps to several lines of print):

```
cp -r /var/mqm/log/DUMQMGR/active   /var/mqm/log/QMGRNAME/active
cp    /var/mqm/log/DUMQMGR/amqhlctl.lfh /var/mqm/log/QMGRNAME/
        ➤  amqhlctl.lfh
```

5 Restart the queue manager:

```
strmqm QMGRNAME
```

The queue manager should restart without a problem. Of course, you lose the data that was on the old logs, but that is better than not having MQ at all.

This procedure is known to work on Sun Solaris and AIX Unix systems but has not been tested on HP/UX or any other version of Unix.

Christine Hills (USA)

MQ news

Candle has made a number of enhancements to its Candle Command Center (CCC) to boost support for MQSeries. New versions of the company's CCC Admin Pac (which ships with MQSeries) and CCC Management Pac support both MQSeries V2.1 for OS/390 and MQSeries V5.1 for other platforms, and a new Management Pac for MQSeries Integrator adds support for MQSeries Integrator Version 2.0. CCC Management Pac for MQSeries Integrator provides tools for MQSeries Integrator configuration and also enables CCC to process and manage MQSeries Integrator-generated events.

Both Admin and Management Pacs are out now, but no details on pricing were released.

For further details contact:
Candle Corp, 2425 Olympic Blvd, Santa Monica, CA 90404, USA
Tel: +1 310 829 5800
Fax: +1 310 582 4287
Web: http://www.candle.com

Candle Ltd, 1 Archipelago, Lyon Way, Frimley, Camberley, Surrey GU16 5ER, UK
Tel: +44 1276 4147000
Fax: +44 1276 414777

* * *

Willow Technology has announced new products to add to its portfolio of clients and servers for MQSeries. In addition to the company's existing MQSeries V2 clients for Data General DG/UX, Silicon Graphics IRIX, and Apple Mac OS, the company has started shipping an MQSeries V2 client for Hewlett-Packard MPE/ix. The new MQSeries V2 server is for SGI IRIX (6.2 or later), which joins existing MQSeries V2 servers for UnixWare (2.1.2 or later) and SCO OpenServer (OSR5.0.2 or later).

The company also announced it has a number of MQSeries V5 clients under development: the IRIX client is now in beta, while beta versions of clients for DG/UX, MacOS, UnixWare, and SCO OpenServer are expected in the near future.

For further information contact:
Willow Technologies Inc, PO Box 320005, Los Gatos, CA 95032, USA
Tel: +1 408 377 7292
Fax: +1 408 377 7293
Web: http://www.willowtech.com

* * *

IBM has announced MQSeries for Tandem NonStop Kernel V2.2.0.1, which has improved performance and scalability and also adds support for more Tandem NonStop Kernel features. The product was developed in conjunction with Tandem, a division of Compaq, and Candle, an IBM business partner. Other features are Euro symbol support and Y2K compliance, and support for both the TCP/IP and SNA protocols.

Out now, no details on pricing were released.

For further information contact your local IBM representative.