46

MQ update

April 2003

In this issue

MQ Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38126
From USA: 01144 1635 38126
Fax: 01635 38345
E-mail: info@xephon.com

North American office

Xephon/QNA
Post Office Box 350100
Westminster CO 80035-0100, USA
Telephone: (303) 410 9344
Fax: (303) 438 0290

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.

Editor

Madeleine Hudson
E-mail: MadeleineH@xephon.com

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 ($33.75) each including postage.

Contributions

When Xephon is given copyright, articles published in *MQ Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. For more information about contributing an article you can download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

MQ Update on-line

Code from *MQ Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at www.xephon.com/mq; you will need to supply a word from the printed issue.

Printed in England.

Hiding WebSphere MQ behind a firewall

INTRODUCTION

The intention of this article is not to explain how a firewall works but to give an overview of some of the services of a firewall. We shall also look at how these services can be used to allow connections between remote WMQ components, using WMQ SupportPac MS81, or, to use its other name, WMQ Internet Pass-Thru (MQIPT).

Using MQIPT a WMQ client or WMQ server can connect to a remote WMQ server across the public network. MQIPT provides a range of options for getting through suitably configured firewalls.

Obviously security is an important issue when traversing the Internet and so MQIPT can be configured to utilize various encryption methods. In the course of this article I will discuss the various features of MQIPT and show how it can be configured.

FIREWALLS

A firewall comprises several different components or services, each designed to control a different aspect of security for various types of network traffic, for example, Telnet, HTTP, SOCKS, etc. These services typically run within the Demilitarized Zone (DMZ).

When a customer installs a firewall they will select those components that suit their own particular needs. A common setup would include an HTTP proxy to allow HTTP traffic out through the firewall and sometimes the firewall will allow incoming HTTP traffic to access an HTTP server inside the firewall.

Most WMQ customers have their own intranets and use a firewall to control access to the public network. The WMQ clients and WMQ servers can easily share and access messages on their local network, but accessing a WMQ server in another network requires the cooperation of both the outgoing and incoming firewall to allow WMQ traffic to flow to the remote location.

A particularly important function of a firewall is to hide the local servers from the outside world to prevent unauthorized access to critical data (see Figure 1).

An alternative solution to using MQIPT in a firewall is to place a WebSphere MQ server in the DMZ of the firewall (see SupportPac MA86 at *http://ibm.com/webspheremq/txppacs/ma86.html* for more information on this subject) and use this as a 'staging post' for WMQ messages. The downside of using this approach is the licensing and administration costs of running another WMQ server and the potential security risk of storing data on disk drives in a server in the DMZ. Another possible solution is the use of a Virtual Private Network (VPN) between the intranets, but this can require dedicated hardware and/or costly leased lines.

MQIPT

MQIPT is designed to be installed in the DMZ to act as a 'tunnel' specifically for WMQ traffic. It will accept a connection request from an WMQ client or WMQ server and route it to the desired destination based on predefined configuration data. Once MQIPT

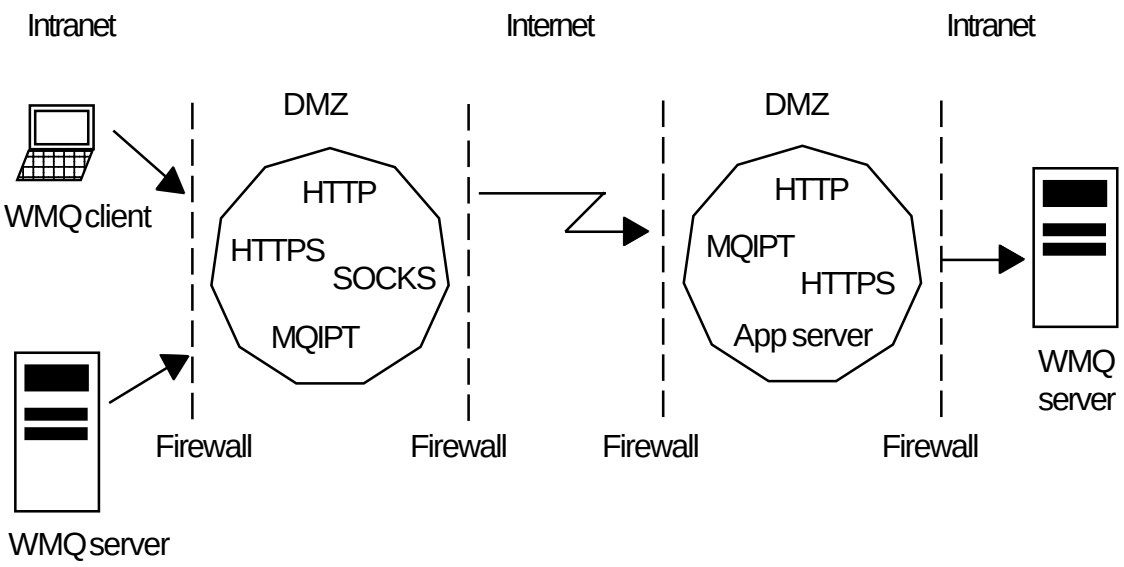

Figure 1: Function and features of a firewall

has established the connection and the handshaking process has completed, WMQ messages are sent and received as on any other WMQ channel connection.

The only change required to WMQ is on the CONNAME of the channel that's being started. It should point to the IP address (or hostname) and port address of the local MQIPT instead of the real destination WMQ server. The local MQIPT is configured to connect to the required destination WMQ server. The WMQ channel (or WMQ client) will not be aware that it is using MQIPT.

To help decide which of the many features of MQIPT should be used within your current WMQ structure it is worth considering the following questions:

- What is the current firewall configuration? Is there:
  - an HTTP proxy?
  - a SOCKS proxy?
  - an application server?
- What type of connections will be made?
  - client to QM?
  - QM to QM?
- For queue manager to queue manager connections, what type of channels will be used?
  - which end will initiate the connection?
- Is encryption required?
  - MQIPT SSL?
  - HTTPS?
- What version of MQ will be used?
- Which platform will MQIPT run on?
  - Windows?
  - AIX?
  - Solaris?
  - Linux?
  - HP?
- Is there an open hole in the firewall just for MQIPT?

– should you limit the number of ports opened for outgoing connections?

The MQIPT book contains many sample configurations and these can be used as a reference when configuring your own MQIPT servers. The MQIPT book can be downloaded from *http://ibm.com/webspheremq/txppacs/ms81.html*.

ADVANTAGES OF USING MQIPT

Detailed below is a summary of the advantages of using MQIPT.

- It has a very small memory footprint; the executable code is less than 1MB.
- MQIPT has minimal hardware requirements, eg Pentium II processor, 128MG RAM, 20MG HDD, or equivalent setup.
- It does not write any user data to disk, thereby improving security and performance.
- It can be configured to use existing firewall services, ie SOCKS, HTTP, and HTTPS.
- It can encrypt data using SSL.
- It can be configured to use specific local port addresses for making outgoing connections.
- It has a servlet version that can be deployed in an application server.
- It's free, has full service support for reporting any problems, and can be downloaded from *http://ibm.com/webspheremq/txppacs/ms81.html*.
- A log file is maintained, showing all connection attempts.
- The Java Security Manager can be used to control connections.
- There is no limit to the number of MQIPT servers that can be chained together.
- One or more MQIPT servers can be administered from a central point, using a GUI-based admin console.

DISADVANTAGES OF USING MQIPT

Detailed below is a summary of the disadvantages of using MQIPT.

- It adds a point of failure in the WMQ channel connection.
- It may depend on the availability and response of an HTTP proxy or SOCKS proxy, etc.
- It requires any HTTP proxy or server used along the connection path to comply with HTTP 1.1 protocols in both directions, ie from the caller and to the destination.
- It adds another component to be administered as part of your messaging infrastructure.
- There must be a second MQIPT in the connection path when using HTTP or SSL tunnelling. In this case, the first MQIPT wraps the data in HTTP or encrypts the data and the second MQIPT unwraps the WMQ data from the HTTP headers or decrypts the data.

PERFORMANCE

Sending data across the Internet introduces certain restrictions on performance and this is sometimes referred to as Network Added Delay (NAD). A single end-to-end connection will involve at least one network service and the availability and performance of these services will be dependent on other network traffic. Sending or receiving large amounts of data over a slow or busy link is likely to highlight this degradation in throughput, just as when using a browser.

Sample performance tests have been run with MQIPT V1.2 to show the overhead of using MQIPT in a simple connection, without using HTTP or SSL, ie no network services were used during the test and all tests were run on an isolated network. Figure 2 shows the network configuration used during the test.

A simple WMQ application was run on the test client to put a WMQ message on a remote queue via a local queue manager (QM). Another WMQ application running on the test server removed the

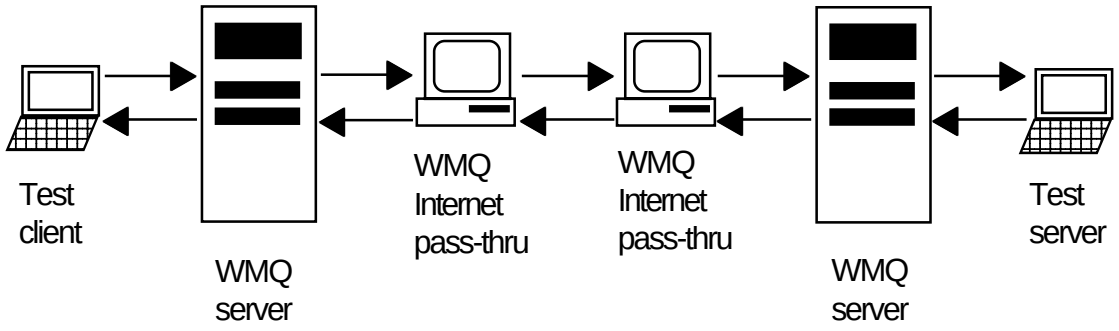

Figure 2: Network configuration used during the performance tests

message from the queue and sent it back to the originator in order to determine the roundtrip time.

A set of benchmark tests was run without using MQIPT in the connection path and then the same tests were run again with two MQIPT servers in the connection, as shown in Figure 2. The overhead of using MQIPT was then calculated.

Most of the overhead of using MQIPT occurs during the initial handshaking process to establish the connection to the target QM. These results exclude this part of the process and show only the overhead of sending WMQ messages of various sizes.

The test client was able to emulate many concurrent users and each set of tests was run with a varying number of clients and different message sizes. Figure 3 shows that when emulating 40 WMQ clients and sending non-persistent messages the overhead of MQIPT is proportional to the size of the message. The figure shows the overhead of a message flowing in one direction through a single MQIPT.

The values given in Figure 3 provide a rough guide to the minimum overhead of using MQIPT; most network services will add a further increase to the overhead. Because every customer configuration is different and the network services used will always vary it is recommended that you perform your own tests to determine the overhead of using MQIPT if this is a significant factor.

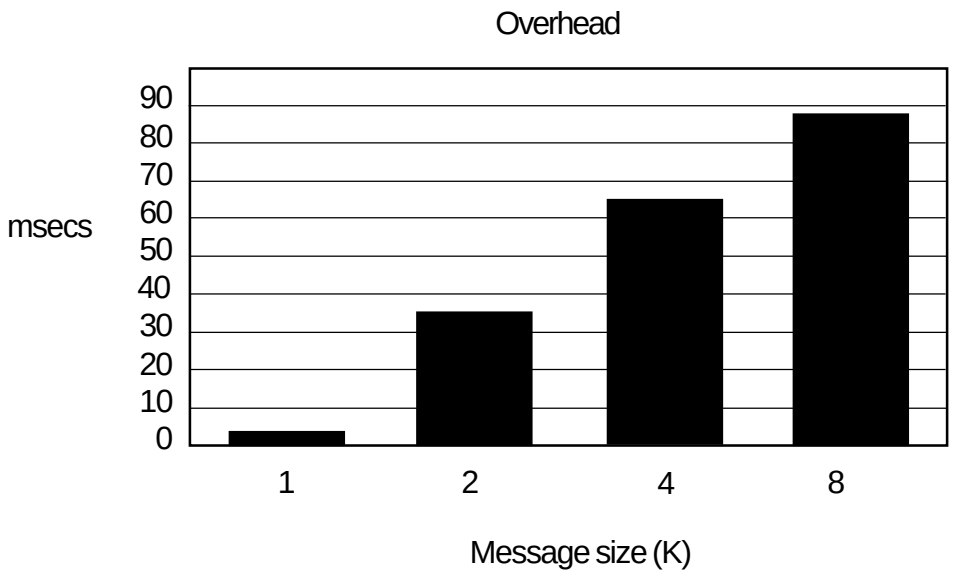

Figure 3: Overhead of a message flowing in one direction through a single MQIPT

SUMMARY

Using MQIPT in the DMZ will give your QM(s) access to the outside world, enable a wide range of inter-enterprise connectivity options, and help to hide your QM(s) from malicious attack.

The performance tests demonstrate that the size of the messages does have an influence on performance and they should be kept as small as possible. This should be an important factor when designing WMQ applications that need to communicate across the public network.

Phil Blake
WebSphere MQ New Technologies
IBM Hursley (UK)

Configuring a Web Client to allow termination of multiple work instances

INTRODUCTION

Have you ever tried to terminate multiple work instances from Workflow using Workflow Web Client? The general interface lets you terminate instances one at a time by clicking the 'Terminate Process Instance' button on the instance list. But suppose you have a development environment comprising 2,000 work instances and you want to delete them all to load new test data. What would you do? With the IBM-supplied Web Client interface you have to delete each instance individually. Let us consider another case. Suppose you have 2,000 work instances and you need to select and delete 100 of them. What would you do? With the given interface you have to use the browser's search option to locate each work instance and delete them individually. You may not have encountered such a situation, but we have. We needed a simple interface so that users could:

- Delete all/multiple work instances with a single mouse-click.
- Select work instances using an input file and delete them with a single mouse-click.

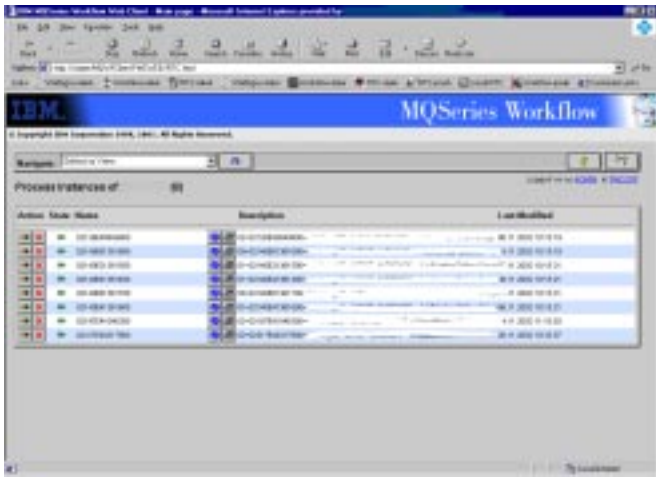

Figure 1: The IBM-supplied Web Client work instances scree

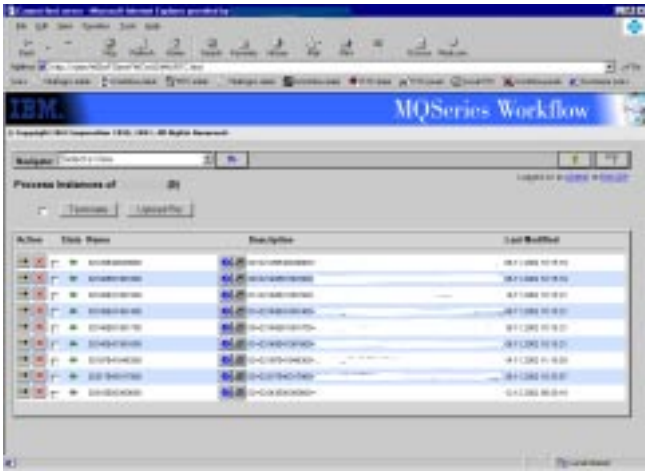

Figure 2: Web Client work instances screen when the LISTViewer.jsp is modified

We came up with the solution that is detailed below. I should warn readers, however, that although this may not necessarily be the most efficient solution it is simple, involves minimal code changes, and is user-friendly.

I would like to thank the IBM MQ Workflow team for providing useful tips. The modifications I have made to *ListViewer.jsp* are intended to help the MQ Workflow development community and it's not my intention to infringe any copyright issues.

SELECTING MULTIPLE WORK INSTANCES

Selecting multiple work instances for terminating

Figure 1 shows the IBM-supplied Web Client work instances screen. This is created by *ListViewer.jsp*, which can be found at *\Program Files\MQSeries Workflow\cfgs\(Your webclient configuration)\WebClient\webpages\forms*. This is the only file that needs to be changed to meet all the above requirements, which is pretty amazing!

In our case, *Name* is a Document Control Number (DCN), which is a unique ID we assign for each work instance. Figure 2 shows

Figure 3: Prompt screen listing names of work instances

the Web Client work instances screen when the *ListViewer.jsp* is modified.

To delete all the instances the user can check the box located at the top left-hand corner, which in turn will select all the checkboxes located beside the 'Delete Process Instance' icon. The checkboxes are created by changing the JSP coding (see code segment #2 and code segment #3) in Appendix A.

To deselect the check boxes the user simply deselects the check

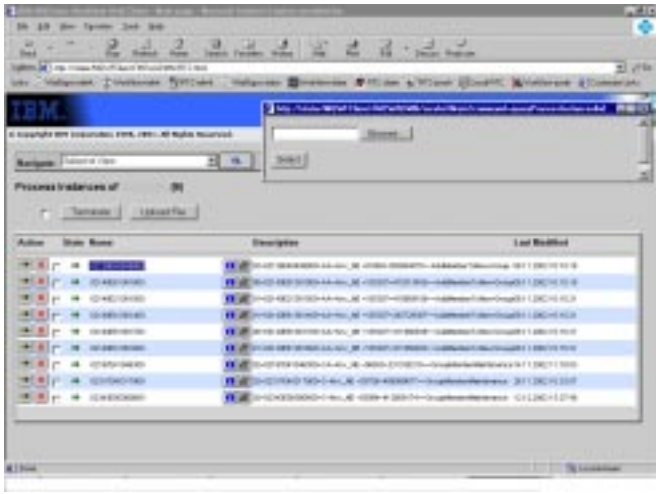

Figure 4: Using the 'upload' file feature

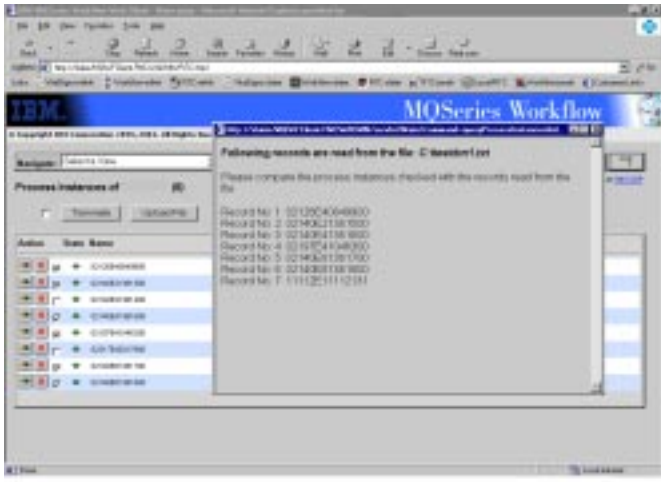

Figure 5: Using the 'upload' file feature

box located at the top left-hand corner. The functionality for doing this is provided in the JavaScript function *checkAllCheckBoxes()* (see code segment #1 in Appendix A). Alternatively, the user can deselect some of the work instances if they choose not to delete them.

After selecting the text boxes the user can terminate the work instances by clicking the terminate button. A prompt screen appears listing the 'Names of the work instances' as shown in Figure 3. This is a prompt for confirmation from the user. If the user doesn't want to delete the instances they can press the 'cancel' button on the pop-up. The functionality for doing this is provided in the JavaScript function *checkAllCheckBoxes()* (see code segment #1).

Select work instances using an input file and terminate

Suppose we need to delete 100 process instances (not in order or sequence) out of a total of 2,000. Instead of searching for and selecting each process instance name, using the 'upload' file feature we could put the process instance names in a text file and programmatically select the matching instances. Figures 4 and 5 illustrate how this feature works.

The user needs to click on the 'Upload File' button to select the

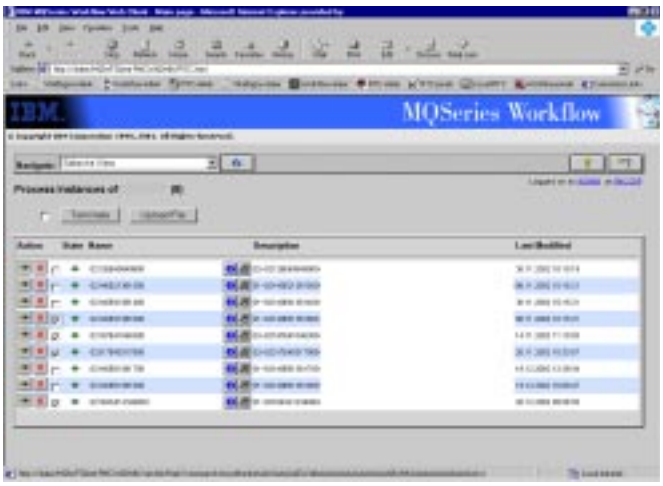

Figure 6: Four instances selected for termination

process instances, using the file located on their local drive. A pop-up window appears, as shown in Figure 4. The user should click on the 'Browse' button and locate the file on the local drive and click on 'select'.

The input file is read and the 'process instances' matching 'input file records' are checked out. A pop-up window appears, asking the user to compare the process instances checked with the records read from the file. There is no button provided to close the window and the user needs to close this window by clicking the 'X'

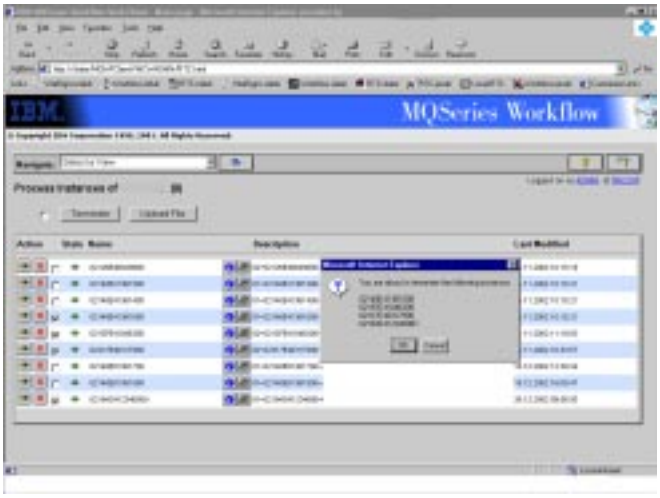

Figure 7: The instances that the user is about to terminate

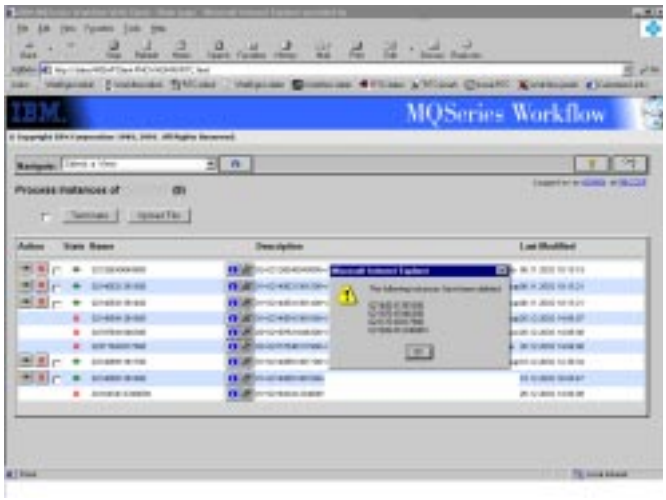

Figure 8: Instances that the user has deleted

located in the top right-hand corner.

The functionality for doing this is provided in the JavaScript functions *checkUsingFileInput()* and *processFile()* (see code segment #4 in Appendix A).

TERMINATING THE INSTANCES AFTER SELECTING

Figure 6 shows that four instances were selected for termination.

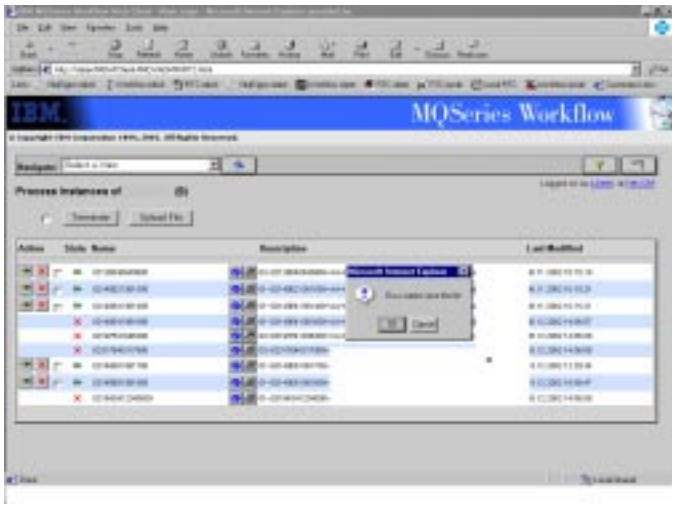

Figure 9: Prompt to save names of terminated instances

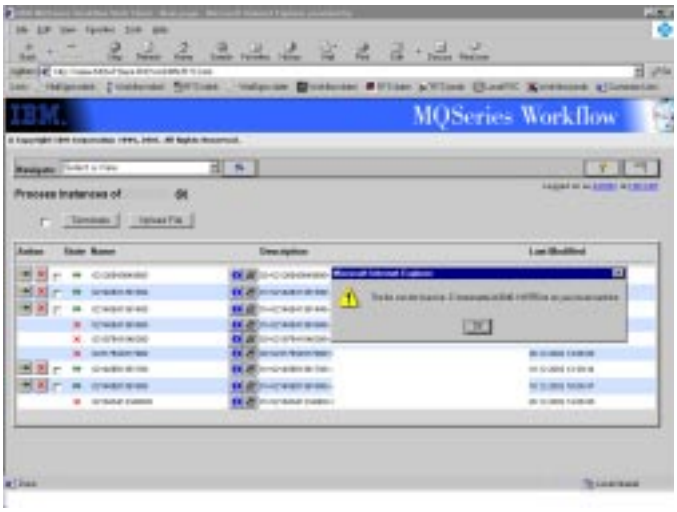

Figure 10: Writing the instance names to a text file

The four instances may be selected either by ‘checking out individual instances’ or ‘using the input file’.

To terminate the selected instances the user should click on the ‘Terminate’ button. A pop-up window appears, as shown in Figure 7, indicating the instances that the user is about to terminate. At this point the user can either cancel the attempt to terminate or click on OK to confirm termination.

A pop-up window appears, as shown in Figure 8, indicating the instances that the user has deleted. Upon acknowledging the notification another pop-up window appears, asking the user whether the names of the terminated instances need to be saved (see Figure 9). This feature helps users keep track of terminated instances. If they need to be saved the instance names will be written to a text file (*termnateList.mmddyy-hhmmss.txt*) created on local drive C:\ (see Figure 10).

The functionality for doing this is provided in JavaScript functions *terminateAllChecked()* (see code segment #5 in Appendix A).

CODE

We have to modify *ListViewer.jsp,* which can be found at *\Program*

Files\MQSeries Workflow\cfgs\(Your webclient configuration) \WebClient\webpages\forms.

Add the code shown in Appendix A, which is italicized, into *ListViewer.jsp* exactly as shown. For clarity I have started each code modification with 'code change #*x* begins' and ended with 'code change #*x* ends'. To explain the functionality I have followed the convention of 'code segments'. I have included the IBM-supplied code to demarcate my code and used ellipses wherever possible.

TROUBLESHOOTING

The easiest way to implement the above changes is to modify *ListViewer.jsp* and restart the Web Client Web application if you are using WebSphere. Taking a back-up of the original *ListViewer.jsp* is strongly recommended. However, this can confuse the 'normal' user, who is not used to the Workflow Web Client screens, with their checkboxes and additional buttons for terminate and delete.

I would suggest that you create an alias Web Client configuration and modify the *ListViewer.jsp* under the alias configuration. Appendix B details the steps required to create the alias Web Client configuration FMCADMIN under WebSphere, using the FMCZUTIL utility.

NOTE

The code is tested in an environment comprising IE, IIS, and WebSphere, with MQSeries Web Client V3.3.0.3.

APPENDIX A

```
<%@ page language="java" contentType="text/html"
errorPage="ViewError.jsp"
   ...  ...  ...
   ...  ...  ...
   ...  ...  ...
  %>
```

```
<html>
...  ...  ...
...  ...  ...
...  ...  ...
<head>
  <title>
    <%if (type == SessionContext.LISTS) {%>
    <%=context.get("ListViewer.Lists")%>  
    <%} else {%>
    <%=context.get("ListViewer.Of", new String[] { distype, list.name()
})%>
    <%}%>
  </title>
  <link rel="stylesheet" type="text/css" href="../webclientstyle.css">
  <style type="text/css">
  <!--
    td
    {
      font-size   :8pt;
      font-family :Arial, Helvetica, sans-serif;
    }
    th
    {
      font-size   :10pt;
      font-weight :bold;
    }
  //-->
  </style>
  <script language='JavaScript'>
// Code change #0 begins
  var termProp1 = new Array();
    var termProp2 = new Array();
  var termProp3 = new Array();
  var numOfTermInst=0;
 // boolean reLoad = false;
//Code change #0 ends
  function fillForm(doc, name, owner, user, cmd, canEmail)
  {
  ...  ...    ...
  ...  ...    ...
  ...  ...    ...
  }
//code change #1 begins
//The following JavaScript function allows the user to check out the
//checkbox located at the left top corner, which in turn would select
//all the checkboxes located beside the "Delete Process Instance" icon.
//To deselect the check boxes, the user simply has to deselect the
check //box located at the left top corner.
// Code Segment #1 begins
```

```
 function checkAllCheckBoxes(){
  //alert ("inst="+document.f2.inst);
    if(document.f2.inst==null){
     document.f1.checkAll.checked = false;
    }
   else{
    if (document.f1.checkAll.checked == true){
      if (document.f2.inst.length==1){
        document.f2.inst.checked=true;
      }
      else{
        for(var i=0;i<document.f2.inst.length;i++)
        {//alert(document.f2.inst[i]);
        document.f2.inst[i].checked=true;}
      }
    }
    else
      if (document.f2.inst.length==1){
        document.f2.inst.checked=true;
      }
      else{
        {
        for(var i=0;i<document.f2.inst.length;i++)
          {document.f2.inst[i].checked=false;}
        }
    }
   }
  }
// Code Segment #1 ends
// Code Segment #5 begins
/* The following function terminates the instances selected and is
executed when terminate button is clicked by the user. The selected
instances ID (generated by workflow) are passed as an array. Each id is
concatenated to another string /MQWFClient-FMCWEB/servlet/
Main?command=terminateInstance&id=ID,to form the URL. The URL so formed
is submitted using GET method using Microsoft.XMLHTTP ActivXObject. The
instance names are concatenated into a text string myalert1 and is used
to write to the local file and to prompt the user. Microsoft FileSystem
Object is used to write the text file. The variable fileName may be
modified by the user to suit his/her requirement.
*/
  function terminateAllChecked(){
    var myAlert1=null;
    var strform=null;
    var url;
    var urlText ;
    var http;
    var check1=true;
    if(document.f2.inst==null){
```

```
      alert("There are NO instances");
      }
      else{
        if (document.f2.inst.length==1){
          if(document.f2.inst.checked==true){
          alert("document.f2.inst.checked "+document.f2.inst.checked);
            if (myAlert1==null){
              myAlert1="\n"+termProp3[0];
              }
            }
                if( confirm("You are about to terminate the following
instance \n"+myAlert1)){
              alert(document.f2.inst.checked);
            if(document.f2.inst.checked==true){
                url = fnSplit(termProp2[i],"this.href =");
              url= fnSplit(url,"'");
    //        url ="/MQWFClient-FMCWEB/servlet/
Main?command=terminateInstance&id="+termProp2[0];
               //url="/MQWFClient-FMCWEB/servlet/
Main?command=terminateInstance&id=UAAAAAEACABiAAAAAAAAAAAAAABAAdAAAAAAAAAAAAAUA%3D%3D"
;
              //alert("url"+url);
              http = new ActiveXObject('Microsoft.XMLHTTP');
              http.open('GET', url, false);
              http.send();
                strform=http.responseText;
            }
            document.write();
           //document.write(strform);
          alert("The following instance has been deleted \n"+myAlert1);
          }
        }
        else{
          for(var i=0;i<document.f2.inst.length;i++){
            if(document.f2.inst[i].checked==true){
              if (myAlert1==null){
                myAlert1="\n"+termProp3[i];
              }
              else{
                myAlert1=myAlert1+"\n"+termProp3[i];
              }
            }
          }
          if( confirm("You are about to terminate the following
instances \n"+myAlert1)){
            try{
              for(var
i=0;((i<document.f2.inst.length)&&(check1==true));i++){
//
```

```
alert("document.f2.inst[i].checked"+document.f2.inst[i].checked+"i="+i+"check1=
"+check1);
                if(document.f2.inst[i].checked==true){
           url = fnSplit(termProp2[i],"this.href =");
           url= fnSplit(url,"'");
                     http = new ActiveXObject('Microsoft.XMLHTTP');
               //post has been replaced with get
               http.open('GET', url, false);
               http.send();
               //alert(http.statusText);
               if(http.statusText!="ok")
                 { alert("resopnse"+http.statusText+"Could terminate
only : "+i);
                 check1=false;}
               strform=http.responseText;
                           }
            }
                document.open();
             document.write(strform);
             document.close();
             alert("The following instances have been deleted
\n"+myAlert1);
             if(confirm("Do u wanto save the list")){
               try{
                 var myDate= new Date();
                   var fileSuffix=
myDate.getDate()+(myDate.getMonth()+1)+ myDate.getYear()+"-
"+myDate.getHours()+myDate.getMinutes()+myDate.getSeconds();
              //If you want to save it diffrent file name or drive pleas
change the variable fileName
               var fileName = "C:\\terminateList"+fileSuffix+".txt";
               var fso = new
ActiveXObject("Scripting.FileSystemObject");
                var file = fso.CreateTextFile(fileName, true);
               file.WriteLine("The following Work Instances have been
deleted on  "+ myDate);
               file.WriteLine(myAlert1);
                  file.Close();
                  alert("The list can be found at :"+fileName+ " on your
local machine");
              }
              catch(e){
                alert("The list could not be written to file"+e);
              }
             }
              }
              catch(e){
               alert("The following instances COULD NOT BE deleted
\n"+e);
```

```
            }
          }
        }
      }
        }
  function fnSplit(string1,delimiter)
{
    var arr;
    var str;
    str = string1;
    arr = str.split(delimiter);
  return arr[1];
}
// Code Segment #5 ends
// Code Segment #4 begins
// Following function allows the user to input the names of the process
// instances using a file. User may browse the local file directories
to
// select the file and then click on the select button to pass the file
// to function processFile()
function checkUsingFileInput(){
   var selectedFileName="my file";
   var strHtml='<html>\n<head>';
   strHtml +='<script type="text/javascript" language="javascript">\n';
       strHtml +='function inputFileName(){\n';
       strHtml +='selectedFileName=document.FileInput.file.value;\n';
  // strHtml +='alert(\"hi\"+selectedFileName);\n';
   strHtml
+='window.opener.processFile(document.FileInput.file.value);\n'
   strHtml +='window.close();';
       strHtml +='}<\/script>\n';
   strHtml=strHtml+'<link rel=\"stylesheet\" type=\"text\/css\"
       href=\"..\/webclientstyle.css\"><\/head>\n<body>\n<form
       name=\'FileInput\' method=\'POST\'>\n';
   strHtml=strHtml+'<input type=\"file\" name=\"file\">\n<\/br><\/br>';
       strHtml=strHtml+'<input type=\"button\" value=\"Select\"';
   strHtml=strHtml+'onClick=\"inputFileName()\">';
      strHtml=strHtml+'\n<\/br><\/br>';
   strHtml=strHtml+'<\/form>\n';
   strHtml=strHtml+'<script type=\"text\/javascript\">\n';
   strHtml=strHtml+'document.FileInput.file.focus()\n';
   strHtml=strHtml+'<\/script>';
   strHtml=strHtml+'<\/body><\/html>';
   var newwindow=
window.open('','SelectFile','width=600,height=100,screenX=100,screenY=0,
toolbar=no,status=no,scrollbars=yes,location=no,menubar=no,directories=no');
   var newdoc=newwindow.document;
   newdoc.open();
   newdoc.write(strHtml);
```

```
   newdoc.close();
   newwindow.focus();
  }
// Following function uses Scripting.FileSystemObject to read the file
// selected. Those of process instances that are matching are checked.
// The records readfrom file are written to a pop-up window to allow
the
// user to compare.Some business specific code that  validates the file
// records has been commented out
  function processFile(fileName) {
    var xFile=fileName;
   // alert( "xFile"+xFile);
    var instNameFromFile = new Array();
    try{
        var fso2= new ActiveXObject("Scripting.FileSystemObject");
    var f2= fso2.OpenTextFile(xFile, 1);
        var numOfFileRecords=0;
    var ctr1=1;
    var fileRecord;
    var pageTitle;
    var strHtml1='<html>\n<head>';
     strHtml1=strHtml1+'<link rel=\"stylesheet\" type=\"text\/css\"
href=\"..\/webclientstyle.css\"><\/head>\n<body>\n<form
name=\'FileInput\' method=\'POST\'>\n';
     strHtml1=strHtml1+'<\/body><\/html>';
    pageTitle="<B> Following records are read from the file: <i>
"+xFile+"<\/i><\/B>";
pageTitle=pageTitle+"  <p> Please compare the process
instances checked with the records read from the file"+"<\/p>";
    var fileWindow=
window.open('','writeFile','width=600,height=400,screenX=100,screenY=0,toolbar=no,status=no,
scrollbars=yes,location=no,menubar=no,directories=no');
    var fileDoc=fileWindow.document;
    fileDoc.open();
    fileDoc.write (pageTitle);
    while(!f2.AtEndOfStream)
    {
      fileRecord=f2.ReadLine();
      instNameFromFile[numOfFileRecords]=fileRecord;
    //  alert("instNameFromFile[numOfFileRecords]"+numOfFileRecords+"
"+instNameFromFile[numOfFileRecords]);
        fileRecord="Record No:
"+ctr1+"  "+fileRecord+"<\/br>";
      fileRecord=strHtml1+fileRecord;
      fileDoc.write (fileRecord);
              ctr1 ++;
      numOfFileRecords ++;
    }
    fileDoc.close();
```

```
    for(var i=0;i<numOfFileRecords;i++){
    //  if(instNameFromFile[i].length == 14){
        for(var j=0;j<=numOfTermInst;j++){
        //    alert("termProp3[j]"+termProp3[j]);
          if(termProp3[j]==instNameFromFile[i]){
            document.f2.inst[j].checked=true;
        }
      }
    //  }
    //  else{
  //  alert(instNameFromFile[i]+" is of invalid length. It should be of
14 digits in length");
    //  }
          }
    }
    catch(e){
      alert(" Failed to read");
      }
}
  // Code Segment #4 ends
  //code change #1 ends
  </script>
</head>
<body>
...  ...  ...
...  ...  ...
...  ...  ...
<!-- Create the heading for the list table ---------------------->
<div class="title">
  <%if (type == SessionContext.LISTS) {%>
    <%=context.get("ListViewer.ListsHeader", String.valueOf(count))%>
 
  <%} else {%>
    <%=context.get("ListViewer.Of", new String[] { distype, list.name()
})%>  (<%=count%>)
  <%}%>
  <br><br>
<!-- code change #2 begins -->
  <form name=f1>
<!--Code segment#2 begins-->
<!--Following code creates a checkbox at the top left corner of Work
Instances list screen, by selecting the check box user may select all
the work instances for deleting.-->
    <%if (type == SessionContext.INSTANCELIST){%>
              <input
type="checkbox" title="check All boxes or uncheck all boxes"
name="checkAll" onClick="checkAllCheckBoxes()">
<!--Code segment#2 ends -->
        <input type="button" name="terminate"
```

```
value="Terminate" title="terminates all checked instances"
onClick="terminateAllChecked()">
        <input type="button" name="uploadfile" value="Upload
File" title="check instances using file input"
onClick="checkUsingFileInput()">
      <%}%>
  </form>
<!-- code change #2 ends -->
</div>
<form name=f2>
<table border="2" bordercolordark="#000000" bordercolorlight="#FFFFFF"
cellspacing="0" cellpadding="5" width="100%" class="listtable">
<tr><td>
<table border="0" cellspacing="0" cellpadding="0" width="100%"
class="listtable">
<tr>
 <th align="left"
width="5%"><%=context.get("ListViewer.Action")%>  </th>
 <%if (type != SessionContext.TEMPLATELIST && type !=
SessionContext.LISTS)
 {%>
   <th align="left" width="1%"
nowrap><b><%=context.get("ListViewer.State")%>  </b></th>
 <%}%>
 <%if (type == SessionContext.LISTS)
 {%><th align="left"
width="10%"><b><%=context.get("ListViewer.KindOfList")%>  </
b></th>
 <%}%>
 <th align="left"
width="20%"><b><%=context.get("ListViewer.Name")%>  </b></th>
 <%if (type != SessionContext.LISTS)
   {%><th align="left" width="5%"> </th> <!-- Properties button --
><%}%>
 <%if (type == SessionContext.WORKLIST) {%>
   <th align="left"
width="30%"><b> <%=context.get("ListViewer.Description")%>  </
b></th>
   <th align="left"
width="15%"><b><%=context.get("ListViewer.ProcessInstance")%>  </
b></th>
 <%} if (type == SessionContext.WORKLIST || type ==
SessionContext.LISTS) {%>
 <th align="left"
width="9%"><b><%=context.get("ListViewer.Owner")%>  </b></th>
 <%} if (type == SessionContext.WORKLIST) {%>
 <th align="left"
width="15%"><b><%=context.get("ListViewer.Received")%>  </
b></th>
```

```
 <%}
   else
   {%>
 <th align="left"
width="30%"><b> <%=context.get("ListViewer.Description")%>  </
b></th>
   <%if (type == SessionContext.INSTANCELIST){%>
 <th align="left"
width="20%"><b><%=context.get("ListViewer.LastModified")%>  </
b></th>
 <%  }
  }%>
</tr>
<!-- Define the Action Icons -->
...  ...  ...
...  ...  ...
...  ...  ...
<!-- Now create the list of instances --------------------------->
<tr><td width="100%" colspan="8"><hr></td></tr>
<!-- code change -->
<% int cnt1=0;
if (type == SessionContext.INSTANCELIST)
    for (int i = 0; i < count; ++i)
    {
      try
      {
        ProcessInstance instance = context.getInstances()[i];
        String oid = instance.persistentOid();
        String name = instance.name();
        Command[] cmds = Command.getActions(instance);
        %>
      <tr class="<%=row[i % row.length]%>">
      <td nowrap><%
      if (cmds.length == 0)
        {%> <%}
      else for (int j = 0; j < cmds.length; ++j)
        {%><%=cmds[j].getTriggerTag(context, oid, name)%>
    <!--code change #3 begins -->
    <b><%//=cmds[j].getCommand()%></b>
      <% if("terminateInstance".equals(cmds[j].getCommand())){%>
    <!--code change #3 begins -->
    <%String valTag="value=\""+cnt1+"\"";
    String myAlert="<script language='JavaScript'> \n termProp1[" +
cnt1 + "]=\""+context+"\";\n"+
    "termProp2[" + cnt1 + "]=\"" +cmds[j].getURL(context,oid,name,true)
+ "\";\n"+"termProp3[" + cnt1 + "]=\"" + name
+"\";\n"+"numOfTermInst="+cnt1+";\n </script>";
    %>
  <!--  <b><%=valTag%></b>-->
```

```
<!-- code segment#3 begins-->
    <input type="checkbox" name="inst" <%=valTag%> >
<!-- code segment#3 ends-->
        <script language='JavaScript'>
        if (document.f2.inst.value==Ø){
        document.f2.inst.length=1;
      }
      </script>
    <%=myAlert%> <% cnt1++; } %>
      <%}%>
      </td>
    <!--code change #3 ends -->
      <td nowrap align="center">
      <img src="../images/state/
<%=context.getPIImageForState(instance.state())%>"
          alt="<%=context.getPIStringForState(instance.state())%>"></
td>
      <td nowrap>
         <%=instance.name()%></td>
      <%context.setInstance(instance);%>
      <td nowrap>
        <a href="<%=context.getCommand("showInstanceProperties",
instance.persistentOid())%>"><%=propImage%></
a><%=context.getTriggerTagFor(SessionContext.INSTANCELIST, context)%></
td>
      <td nowrap>
         <%=context.null2Empty(instance.description())%>
      </td>
      <td nowrap>
       <%=context.toString(instance.lastModificationTime())%> </
td>
      </tr>
    <%}
      catch(FmcException xcpt) {
       if (xcpt.rc != FmcException.FMC_ERROR_DOES_NOT_EXIST) throw
xcpt;
      }
    } /* End of for ----------- Process Instances ----------- */
...  ...  ...
...  ...  ...
...  ...  ...
  <%} /* End of for ----------- Process Templates ----------- */
...  ...  ...
...  ...  ...
...  ...  ...    } /* End of for ----------- Work Items ----------- */
  }
...  ...  ...
...  ...  ...
...  ...  ...  } /* End of for ----------- List of Lists ----------- */
```

```
  } %>
</table>
</form>
<!-- code change #5 -->
</td></tr></table>
</body></html>
```

APPENDIX B

Creating an alias Web Client configuration:

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
C:\Documents and Settings\db2admin>cd \
C:\>fmczutil
   FMC3201I Configuration Commands Menu:
     l ... List
     s ... Select
     c ... Create
     d ... Change default configuration
     x ... Exit Configuration Commands Menu
c
     Configuration identifier : [FMC] FMCWADMN
   FMC3210I Select Category Menu:
     s ... ( ) Server
     i ... ( ) Runtime Database Utilities
     b ... ( ) Buildtime
     c ... ( ) Client with queue manager
     j ... ( ) Java Agent
     w ... ( ) Web Client
     a ... all
     n ... none
     x ... Exit Select Category Menu
w
   FMC3210I Select Category Menu:
     s ... ( ) Server
     i ... ( ) Runtime Database Utilities
     b ... ( ) Buildtime
     c ... ( ) Client with queue manager
     j ... ( ) Java Agent
     w ... (X) Web Client
     a ... all
     n ... none
     x ... Exit Select Category Menu
j
   FMC3210I Select Category Menu:
     s ... ( ) Server
     i ... ( ) Runtime Database Utilities
```

```
     b ... ( ) Buildtime
     c ... (A) Client with queue manager
     j ... (X) Java Agent
     w ... (X) Web Client
     a ... all
     n ... none
     x ... Exit Select Category Menu
x
-  Configuration of queue manager ...
     System group name        : [FMCGRP1] FMCGRP
     System name              : [FMCSYS1] FMCSYS
     Queue manager name       : [FMCCONQM] FMCCONQM
     Queue prefix             : [FMC] FMC
-  Configuration of client ...
     Channel definition table file         : [d:\program files\mqseries
workflow
\chltabs\mqwfchl.tab]
-  Configuration of Java Agent ...
-  FMC33749I Selected Locator Policy : Local bindings

   FMC336Ø6I Specify information about garbage collection (reaper) ...:
     Agent cycle (in seconds)             : [3ØØ]
     Client threshold (number of objects) : [1ØØØ]
     Client cycle (in % of agent cycle)   : [9Ø]
-  Configuration of Web Client ...
   FMC33942I Specify the root URI of the Web Client :
     Root URI : [MQWFClient-FMCWADMN]
   FMC33777I Select application server ...:
     w ... ( ) WebSphere 3.x
     f ... (X) WebSphere 4.Ø (EAR)
     o ... ( ) Other
     j ... ( ) Other (Servlet 2.2 / J2EE 1.2)
w
     Code Version                         : [33Ø3]
   FMC336Ø7I Specify information about the WebSphere Application Server
...:
     Installation directory          : [d:\WebSphere\AppServer]
     TCP/IP address of administration node : [slater]
     TCP/IP address of name service host   : [slater]
     TCP/IP port number of name service    : [9ØØ]
     XML configuration skeleton file name  : [fmcoh354.skel]
     c ... Create configuration profile for 'FMCWADMN' now
     s ... Save input to file
     r ... Review/change input
     x ... Exit (input for configuration 'FMCWADMN' will be lost)
-  FMC3368ØI The profile for the configuration 'FMCWADMN' was updated
successfully.
-  Do you want to configure the Web Client within the WebSphere
Application Server now?
     y ... Yes
```

```
     n ... No
y
[Ø2.12.1Ø 17:41:14:5ØØ EST] ce421533 NodeConfig    A XMLC0053I:
Importing Node :
 slater
[Ø2.12.1Ø 17:41:14:671 EST] ce421533 ApplicationSe A XMLC0053I:
Importing ApplicationServer : MQWF Web Client - FMCWADMN
[Ø2.12.1Ø 17:41:15:671 EST] ce421533 ServletEngine A XMLC0053I:
Importing ServletEngine : Servlet Container
[Ø2.12.1Ø 17:41:17:Ø62 EST] ce421533 WebApplicatio A XMLC0053I:
Importing WebApplication : MQWFClient-FMCWADMN
[Ø2.12.1Ø 17:41:17:562 EST] ce421533 ServletConfig A XMLC0053I:
Importing Servlet : ErrorReporter
[Ø2.12.1Ø 17:41:17:875 EST] ce421533 ServletConfig A XMLC0053I:
Importing Servlet : file
[Ø2.12.1Ø 17:41:18:281 EST] ce421533 ServletConfig A XMLC0053I:
Importing Servlet : jsp11
[Ø2.12.1Ø 17:41:18:546 EST] ce421533 ServletConfig A XMLC0053I:
Importing Servlet : Main
[Ø2.12.1Ø 17:41:18:89Ø EST] ce421533 SessionManage A XMLC0053I:
Importing SessionManager : Session Manager
[Ø2.12.1Ø 17:41:19:281 EST] ce421533 ApplicationSe A XMLC0053I:
Importing ApplicationServer : MQWF Web Client - FMCWADMN
[Ø2.12.1Ø 17:41:19:687 EST] ce421533 ContainerConf A XMLC0053I:
Importing Container : Default Container
   FMC332Ø1I Configuration Commands Menu:
     l ... List
     s ... Select
     c ... Create
     d ... Change default configuration
     x ... Exit Configuration Commands Menu
x
C:\>
```

Chandra Upadhyayula
Programmer Analyst (USA)

MQ Telnet interface for OS/390

I was recently approached by someone in management and asked to teach an MQ class to a bunch of Java programmers. While MQ is a nice multi-platform tool the goal was to have Java applications directly access queues on OS/390. Even though this is not a

problem for MQ our Java programmers had no prior experience with OS/390. I was told I had to teach them how to be self-sufficient in one day. As straightforward as MQ is, one day was still a challenge! This meant I had to impart MQ concepts, MQ configuration, queue administration, queue manipulation, programming techniques, and testing techniques, as well as basic TSO/ISPF and JCL concepts. Fortunately, these programmers were seasoned and experienced – just lacking OS/390 experience.

Since we already had several Java applications using MQ this was actually the least of my concerns since we had so many working examples and access to the original programmers. My biggest concern was removing the TSO/ISPF and JCL/JES2/SDSF learning curve. All the programmers were well-versed in multiple flavours of Unix and Microsoft and had extensive Telnet experience. Since I had been doing a significant amount of work with OS/390 Unix Systems Services (USS), it occurred to me that it might be possible to write a Telnet interface to MQ similar to that found on the other platforms (kind of like RUNMQSC).

With this idea in mind I proposed a small development project to write this interface to minimize the learning curve and improve the time to market. The idea was accepted and the resulting set of five REXX EXECs allows anyone with authority and a Telnet session to the USS side of OS/390 to work with the entire MQ command set and directly manipulate queues.

This required REXX EXECs on the traditional MVS side and REXX EXECs on the USS side (see Table 1). The interface allows for the incorporation of homegrown GET and PUT utilities. The version provided includes my MQGET and MQPUT routines, which use the free MQ MA18 SupportPac (found at *http://www-3.ibm.com/software/ts/mqseries/txppacs/ma18.html*).

The first REXX EXEC is called RUNMQSC. This runs from the USS side. Place this routine in any USS directory and make the path known to the users or place it in a directory in the users' PATH. This routine provides the basic menu of features. It will first prompt for the QMGR to connect to, then it will present the menu in typical Telnet 'scroll and roll' format. Maybe someday I'll rewrite it to use

REXX EXEC	Description	Environment
RUNMQSC	Menu driver	USS
MVSREXX	USS to MVS interface	USS
MQUTIL	CSQUTIL wrapper	MVS
MQGET	MQ browse tool (uses MA18)	MVS
MQPUT	MQ put tool (uses MA18)	MVS

Table 1: Required REXX EXECs

a Telnet screen formatter. Here is a quick view of what it looks like:

```
RZENUK:/u/cyclone/bin: >runmqsc
Unix Systems Services MQ Interface - brought to you by Banisco
Enter the QMGR name or press ENTER for the CSQ1 default
1) Display all queues
2) Display a specific queue and all details
3) Create a new queue
4) Empty an existing queue
5) Delete an existing queue
6) Enter any valid MQ command
7) Browse the contents of a queue
8) Manually enter a message into a queue
9) Load a single message into a queue from a file
Select a valid option 1 - 9 and press enter
```

The RUNMQSC REXX EXEC will format the appropriate commands then call another REXX EXEC called MVSREXX to execute REXX EXECs on the MVS side. The following portion on RUNMQSC will need to be tailored for your environment.

```
/* Defaults                                                    */
EXITRC    = Ø                   /* Default exit code          */
defqmgr   = 'CSQ1'              /* Default QMGR               */
mvsrexx   = 'mvsrexx'           /* USS EXEC to launch MVS EXEC's  */
execdsn   = 'sys1.local.exec'   /* SYSEXEC DSN                */
mqutil    = 'MQUTIL'            /* EXEC to execute CSQUTIL    */
mqget     = 'MQGET'             /* EXEC to execute MQGET      */
mqput     = 'MQPUT'             /* EXEC to execute MQPUT      */
defqmodel = 'cyclone.model'     /* MQ Queue Model for new queues  */
```

The following values will most likely need to change in your environment:

- DEFQMGR
  - the default QMGR in your environment.
- EXECDSN
  - the PDS you put MQUTIL, MQGET, and MQPUT into.
- DEFQMODEL
  - the default queue to model after.

The MVSREXX REXX EXEC addresses the Shell, allocates a SYSEXEC, and issues the USS TSO command to pass a TSO request to the MVS side. Not all TSO commands are supported using this technique but everything necessary for this interface works. MVSREXX passes requests to MVS for three REXX EXECs to support all nine options shown above. MQUTIL is a REXX wrapper for CSQUTIL. MQGET provides a queue browse function and MQPUT provides a single message put function.

MQUTIL supports options one to six (see Table 2). MQUTIL also removes several messages from CSQUTIL output to make the output easier to work with in a Telnet session. MQGET is option seven and MQPUT options eight and nine.

MQGET is a simple MQ non-destructive browse REXX EXEC based on MA18. This can be replaced easily in the defaults section of RUNMQSC with a local alternative. Simply wrap the local browse program in a REXX EXEC and identify it as the MQGET EXEC. The parameters passed to MQGET are QMGR and QNAME. The optional QOTRUNC is available to increase the default output truncation from 2,000 bytes. MQGET also works standalone on the MVS side.

MQPUT is a simple MQ PUT REXX EXEC also based on MA18. This can also be replaced using the same technique described for MQGET. MQPUT can be used to PUT any string as a message or use any MVS sequential dataset or a file in an HFS directory as input. MQPUT will only do a single put. The file option was created since we have some fairly large message sizes that are tedious to type (digital certificates too). All messages are PUT with the MQFMT_STRING option to support cross-platform messaging.

Option	Command issued
1	DISPLAY QUEUE(*)
2	DISPLAY QUEUE("'qname"') ALL
3	DEFINE QLOCAL("'qname"') DESCR("'qdesc"') LIKE("'qmodel"')
4	EMPTY QUEUE('qname')
5	DELETE QLOCAL("'qname"')
6	Any valid CSQUTIL command i.e DISPLAY THREAD(*)
7	MQGET
8	MQPUT text from screen
9	MQPUT contents of a file/dataset

Table 2: List of options and commands

MQPUT also works standalone on the MVS side.

RUNMQSC

```
/*                                REXX                                  */
/* Purpose: Execute CSQUTIL from a telnet session                       */
/* Syntax: runmqsc qmgr mqcmd                                           */
/* Parms: qmgr       - The MVS QMGR to connect to (default CSQ1)        */
/*        mqcmd      - MQ command to execute (prompts if missing)       */
/*                        Change Log                                    */
/*  Author       Date      Reason                                       */
/*  R. Zenuk    Oct 2002   Initial Creation                             */
/*  R. Zenuk    11/07/02   Minor tweaks                                 */
/* Welcome                                                              */
 say 'Unix Systems Services MQ Interface - brought to you by Banisco'
 say
/* Defaults                                                            */
 EXITRC    = 0                    /* Default exit code                 */
 defqmgr   = 'CSQ1'               /* Default QMGR                      */
 mvsrexx   = 'mvsrexx'            /* USS EXEC to launch MVS EXEC's     */
 execdsn   = 'sys1.local.exec'    /* SYSEXEC DSN                       */
 mqutil    = 'MQUTIL'             /* EXEC to execute CSQUTIL           */
 mqget     = 'MQGET'              /* EXEC to execute MQGET             */
```

```
 mqput     = 'MQPUT'               /* EXEC to execute MQPUT          */
 defqmodel = 'cyclone.model'       /* MQ Queue Model for new queues  */
/* Accept QMGR and MQCMD                                             */
 parse arg qmgr mqcmd
/* If QMGR is missing, prompt for it                                 */
 if qmgr = '' then
    do
     say
     say 'Enter the QMGR name or press ENTER for the' defqmgr 'default'
     say
     pull qmgr
     if qmgr = '' then qmgr = defqmgr
    end
 else
    do
     qmgr = translate(qmgr)
    end
/* If MQCMD is included execute and get out                          */
 if mqcmd <> '' then
    do
     mvsrexx execdsn mqutil qmgr '"'mqcmd'"'
     exit(RC)
    end
/* If MQCMD is missing, display a menu                               */
 menu: nop
 do forever
    say
    say '1) Display all queues'
    say '2) Display a specific queue and all details'
    say '3) Create a new queue'
    say '4) Empty an existing queue'
    say '5) Delete an existing queue'
    say '6) Enter any valid MQ command'
    say '7) Browse the contents of a queue'
    say '8) Manually enter a message into a queue'
    say '9) Load a single message into a queue from a file'
    say
    say 'Select a valid option 1 - 9 and press enter'
    pull choice
    if pos(choice,'123456789') = choice then leave
 end
/* Procesing options                                                 */
 select
/* Display all Queues                                                */
   when choice = 1 then
        do
         say
         mvsrexx execdsn mqutil qmgr '"DISPLAY QUEUE(*)"'
        end
/* Display a specific queue                                          */
   when choice = 2 then
```

```
        do
         say
         say 'Please enter the Queue to display (case is important)'
         say
         parse pull qname
         say
         qname = qname || '*'
         mvsrexx execdsn mqutil qmgr '"DISPLAY QUEUE('''qname''') ALL"'
        end
/* Create a new queue                                                */
   when choice = 3 then
        do
         say
         say 'Please enter the Queue name to create (case is
important)'
         say
         parse pull qname
         say
         say 'Please enter a description for the queue'
         say
         parse pull qdesc
         say
         say 'Please enter a queue to clone or enter to use' defqmodel
         say
         parse pull qmodel
         if qmodel = '' then qmodel = defqmodel
         say
         mvsrexx execdsn mqutil qmgr '"DEFINE QLOCAL('''qname''')"',
               '"DESCR('''qdesc''') LIKE('''qmodel''')"'
        end
/* Empty a queue                                                     */
   when choice = 4 then
        do
         say
         say 'Please enter the Queue name to empty (case is important)'
         say
         parse pull qname
         say
         say 'Are you sure you want to clear' qname '(enter NO to
stop)'
         say
         pull qsure
         say
         if substr(qsure,1,1) = 'N' then
            do
             say qname 'will not be cleared'
             signal shutdown
            end
         say
         mvsrexx execdsn mqutil qmgr '"EMPTY QUEUE('qname')"'
        end
```

```
/* Delete a queue                                                    */
   when choice = 5 then
        do
         say
         say 'Please enter the Queue name to delete (case is
important)'
         say
         parse pull qname
         say
         say 'Are you sure you want to delete' qname '(enter NO to
stop)'
         say
         pull qsure
         say
         if substr(qsure,1,1) = 'N' then
            do
             say qname 'will not be deleted'
             signal shutdown
            end
         say
         mvsrexx execdsn mqutil qmgr '"DELETE QLOCAL('''qname''')"'
        end
/* Enter any MQ command                                              */
   when choice = 6 then
        do
         say
         say 'Please enter any valid MQ command'
         say
         parse pull mqcmd
         mvsrexx execdsn mqutil qmgr '"'mqcmd'"'
        end
/* Run MQGET to display the contents of a Queue                      */
   when choice = 7 then
        do
         say
         say 'Please enter the Queue name to browse (case is
important)'
         say
         parse pull qname
         mvsrexx execdsn mqget qmgr qname
        end
/* Run MQPUT to insert new messages by hand                          */
   when choice = 8 then
        do
         say
         say 'Please enter the Queue name to load (case is important)'
         say
         parse pull qname
         say
         say 'Please enter the message text'
         say
```

```
         parse pull message
         mvsrexx execdsn mqput qmgr qname message
        end
/* Run MQPUT to insert the contents of a file                          */
   when choice = 9 then
        do
         say
         say 'Please enter the Queue name to load (case is important)'
         say
         parse pull qname
         say
         say 'Please enter the MVS dataset or HFS file to load'
         say
         parse pull file
         mvsrexx execdsn mqput qmgr qname word(file,1)
        end
   otherwise say 'Unexpected option, please hang up and dial again'
 end
/* Shutdown                                                           */
 shutdown: say
           say "Enter 'Q' to quit, press ENTER to continue"
           pull stop
           if stop <> 'Q' then signal menu
           exit(EXITRC)
```

MVSREXX

```
/*                              REXX                                  */
/* Purpose: Execute an MVS REXX EXEC from the USS side                */
/* Syntax: mvsrexx dsn mem parms                                      */
/* Parms: dsn        - REXX EXEC PDS to allocate to SYSEXEC           */
/*        mem        - The REXX EXEC to execute                       */
/*        parms      - Parms to pass to the EXEC                      */
/*                     Change Log                                     */
/*  Author       Date      Reason                                     */
/*  R. Zenuk    Oct 2002   Initial Creation                           */
/* Accept DSN, MEM and PARMS                                          */
 parse arg dsn mem parms
 dsn = translate(dsn)
 mem = translate(mem)
/* Display a startup message                                          */
 say 'EXEC:' mem 'executing from:' dsn 'using parms:' parms
/* Address the SHELL to ALLOC SYSEXEC and execute the TSO command    */
 address 'SH' 'TSOALLOC=SYSEXEC',
         'SYSEXEC="ALLOC DSN(''''dsn'''') SHR REUSE MSG(2)"',
         'tso "%'mem parms'"'
 EXITRC = RC
/* Display a shutdown message                                         */
 shutdown: say 'EXEC:' mem 'executed from:' dsn 'RC='EXITRC
           exit(EXITRC)
```

MQUTIL

```
/*                                REXX                                  */
/* Purpose: Run CSQUTIL                                                 */
/* Syntax:  Run CSQUTIL and accept commands                             */
/* Parms: qmgr       - Queue Manager to attach to                       */
/*        cmd        - Command to execute                               */
/*                        Change Log                                    */
/*  Author       Date      Reason                                       */
/*  R. Zenuk    Oct 2002   Initial Creation                             */
/*  R. Zenuk    11/05/02   Trimmed all blanks off incoming command      */
/*  R. Zenuk    11/07/02   Fixed parse problem and improved output      */
/* Accept QMGR and CMD parms                                            */
 parse arg qmgr cmd
 if qmgr = '' then
    do
     say 'QMGR is missing'
     exit(10)
    end
 qmgr = translate(qmgr)
 if cmd = '' then
    do
     say 'MQ command is missing'
     exit(11)
    end
/* Allocate all required DD's                                           */
"ALLOC F(SYSPRINT) UNIT(VIO) SPACE(1 5) CYLINDERS"
 EXITRC = RC
 if EXITRC <> 0 then say 'ALLOC SYSPRINT error RC='EXITRC
"ALLOC F(SYSIN) UNIT(VIO) SPACE(1 1) TRACKS LRECL(80) BLKSIZE(0)"
 EXITRC = RC
 if EXITRC <> 0 then say 'ALLOC SYSIN error RC='EXITRC
"ALLOC F(CMDINPUT) UNIT(VIO) SPACE(1 1) TRACKS LRECL(80) BLKSIZE(0)"
 EXITRC = RC
 if EXITRC <> 0 then say 'ALLOC CMDINPUT error RC='EXITRC
/* Prepare SYSIN for Command Input (redirect to CMDINPUT) unless        */
/* this is a supported native CSQUTIL utility function                  */
 cmd = strip(cmd)
 if word(cmd,1) = 'EMPTY' then
    sysin.1 = cmd
 else
    sysin.1 = 'COMMAND DDNAME(CMDINPUT)'
"EXECIO * DISKW SYSIN (STEM SYSIN. FINIS"
 EXITRC = RC
 if EXITRC <> 0 then say 'EXECIO SYSIN error RC='EXITRC
/* Determine if clause parsing is required                              */
 if length(cmd) >= 75 then
    do c=1 to words(cmd)
       if c < words(cmd) then
          cmdinput.c = word(cmd,c) '+'
       else
```

```
          cmdinput.c = word(cmd,c)
    end
 else
    cmdinput.1 = cmd
/* Load the MQ command (one clause at a time if cmd length >= 75)    */
"EXECIO * DISKW CMDINPUT (STEM CMDINPUT. FINIS"
 EXITRC = RC
 if EXITRC <> Ø then say 'EXECIO CMDINPUT error RC='EXITRC
/* Call CSQUTIL                                                      */
 address ATTCHMVS "CSQUTIL" "QMGR"
 UTILRC = RC
 if UTILRC <> Ø then say 'CSQUTIL error RC='UTILRC
/* Read the output from SYSPRINT                                     */
"EXECIO * DISKR SYSPRINT (STEM SYSPRINT. FINIS"
 EXITRC = RC
 if EXITRC <> Ø then say 'EXECIO SYSPRINT error RC='EXITRC
/* Filter out the extraneous lines from SYSPRINT                     */
 do i=1 to sysprint.Ø
    select
      when word(sysprint.i,1) = '1CSQM4Ø1I' then iterate
      when word(sysprint.i,1) = 'CSQM4Ø1I'  then iterate
      when word(sysprint.i,1) = 'CSQM4Ø2I'  then iterate
      when word(sysprint.i,1) = 'CSQM4Ø3I'  then iterate
      when word(sysprint.i,1) = 'CSQM4Ø6I'  then iterate
      when word(sysprint.i,1) = 'ØCSQN2Ø5I' then iterate
      when word(sysprint.i,2) = 'CSQUØØØI'  then iterate
      when word(sysprint.i,2) = 'CSQUØØ1I'  then iterate
      when word(sysprint.i,2) = 'CSQUØØ5I'  then iterate
      when word(sysprint.i,2) = 'CSQUØ55I'  then iterate
      when word(sysprint.i,2) = 'CSQUØ57I'  then iterate
      when word(sysprint.i,2) = 'CSQUØ58I'  then iterate
      when word(sysprint.i,2) = 'CSQU12ØI'  then iterate
      when word(sysprint.i,2) = 'CSQU122I'  then iterate
      when word(sysprint.i,2) = 'CSQU127I'  then iterate
      when word(sysprint.i,2) = 'CSQU133I'  then iterate
      when word(sysprint.i,2) = 'CSQU14ØI'  then iterate
      when word(sysprint.i,2) = 'CSQU142I'  then iterate
      when word(sysprint.i,2) = 'CSQU143I'  then iterate
      when word(sysprint.i,2) = 'CSQU144I'  then iterate
      when word(sysprint.i,2) = 'CSQU148I'  then iterate
      when word(sysprint.i,1) = 'COMMAND '  then iterate
      when pos('QUEUE(SYSTEM.',sysprint.i) <> Ø then iterate
      when cmd = 'DISPLAY QUEUE(*)' & pos('TYPE',sysprint.i) <> Ø then
           iterate
      otherwise say strip(sysprint.i)
    end
 end
/* Free all files                                                    */
"FREE F(SYSIN)"
"FREE F(CMDINPUT)"
```

```
"FREE F(SYSPRINT)"
/* Shutdown                                                          */
 shutdown: if UTILRC > EXITRC then
              exit(UTILRC)
           else
              exit(EXITRC)
```

MQGET

```
/*                                REXX                               */
/* Purpose: Simple MQ GET program to print a Q                       */
/* Syntax:  MQGET qmgr qname qotrunc                                 */
/* Parms: qmgr       - Q Manager to Connect to                       */
/*        qname      - Q Name                                        */
/*        qotrunc    - Q Output truncation limit                     */
/*                        Change Log                                 */
/*  Author       Date      Reason                                    */
/*  R. Zenuk    Feb 2001   Initial Creation                          */
/*  R. Zenuk    05/31/01   Fixed comment                             */
/*  R. Zenuk    10/10/02   Upgraded for USS Telnet support           */
/* Accept message                                                    */
 parse arg qmgr qname qotrunc
 if length(qmgr) <> 4 then
    do
     say 'QMGR "'qmgr'" is not correct, try something like CSQ1'
     exit(10)
    end
 if qname  = '' then
    do
     say 'QNAME is missing (remember case is important)'
     exit(11)
    end
 if qotrunc = '' then qotrunc = 2000
/* Initialize the API                                                */
 mqrc = RXMQV('INIT')
 if word(mqrc,1) <> 0 then say mqrc
/* Connect to a QMGR                                                 */
 mqrc = word(RXMQV('CONN', qmgr),1)
 if mqrc <> 0 then
    do
     say 'MQCONN to QMGR "'qmgr'" failed MQRC='mqrc
     exit(mqrc)
    end
/* Open the Queue                                                    */
 options = mqoo_inquire+mqoo_output+mqoo_browse+mqoo_set
 mqrc = word(RXMQV('OPEN', qname, options , 'h', 'ood.' ),1)
 if mqrc <> 0 then
    do
     say 'MQOPEN for QUEUE "'qname'" failed MQRC='mqrc
     exit(mqrc)
```

```
   end
/* Inquire on the number of messages on the Q                        */
 atrin = mqia_current_q_depth
 atrou = ''
 mqrc = RXMQV('INQ', h, atrin, 'atrou' )
 if word(mqrc,1) <> Ø then say mqrc
 say 'QMGR:' qmgr 'Q Name:' qname 'Q Depth:' atrou
/* Get and print all the messages                                    */
 do i=1 to atrou
    msg.Ø = qotrunc
    msg.1 = ''
    igmo.opt = MQGMO_WAIT+MQGMO_BROWSE_NEXT+MQGMO_CONVERT
    igmd.ENC  = MQENC_NATIVE
    igmd.CCSI = MQCCSI_Q_MGR
    ogmd.ENC  = MQENC_NATIVE
    ogmd.CCSI = MQCCSI_INHERIT
    mqrc = RXMQV('GET', h,'msg.','igmd.','ogmd.','igmo.','ogmo.')
    if word(mqrc,1) <> Ø then say mqrc
    say '===> MSG' i 'Length:' msg.Ø 'Msg Text:' msg.1
 end
/* Close the Q                                                       */
 mqrc = RXMQV('CLOSE', h, mqco_none)
 if word(mqrc,1) <> Ø then say mqrc
/* Disconnect from the QMGR                                          */
 mqrc = RXMQV('DISC', )
 if word(mqrc,1) <> Ø then say mqrc
/* Terminate the API                                                 */
 mqrc = RXMQV('TERM', )
 if word(mqrc,1) <> Ø then say mqrc
```

MQPUT

```
/*                                REXX                               */
/* Purpose: Simple MQ PUT program                                    */
/* Syntax:  MQPUT qmgr qname file                                    */
/* Parms: qmgr       - Q Manager to Connect to                       */
/*        qname      - Q Name                                        */
/*        message    - Any text or sequential filename               */
/*                        Change Log                                 */
/*  Author       Date      Reason                                    */
/*  R. Zenuk    Feb 2ØØ1   Initial Creation                          */
/*  R. Zenuk    Ø6/14/Ø1   Added file support                        */
/*  R. Zenuk    Ø6/19/Ø1   Added MQFMT_STRING support                */
/*  R. Zenuk    Ø6/19/Ø1   Combined MQPUT and MQPUTF                 */
/*  R. Zenuk    Ø9/24/Ø2   Fixed the quoting problem for files       */
/*  R. Zenuk    1Ø/1Ø/Ø2   Upgraded for USS Telnet support           */
/*  R. Zenuk    1Ø/29/Ø2   Added HFS file support                    */
/* Accept message                                                    */
 parse upper source . . execname . execdsn . . execenv .
 parse arg qmgr qname message
```

```
 if length(qmgr) <> 4 then
    do
     say 'QMGR' qmgr 'does not appear correct, try something like CSQ1'
     exit(1Ø)
    end
 if qname  = '' then
    do
     say 'QNAME is missing (remember case is important)'
     exit(11)
    end
 if message = '' then
    do
     say 'The message or file to MQPUT is missing'
     exit(12)
    end
/* If the message is a single word wrap it in quotes for SYSDSN and  */
/* set the text variable to NO                                       */
 if words(message) = 1 then
    do
     text = 'NO'
     catcheck = "'"message"'"
    end
/* If the message is not a single word, assume this is the message   */
/* set the text variable to YES                                      */
 else
    do
     text = 'YES'
    end
/* If the message begins with '/', assume an HFS file.  If it passes */
/* the SYSDSN check, assume MVS DSN                                  */
 select
   when substr(message,1,1) = "/" then file = word(message,1)
   when sysdsn(catcheck) = 'OK' then file = word(message,1)
   when sysdsn(catcheck) <> 'OK' then text = 'YES'
   otherwise text = 'YES'
 end
/* If the text = NO this is a file to process                        */
 if text = 'NO' then
    do
/* Read the HFS file                                                 */
     if substr(file,1,1) = '/' then
        do
         tdate = date('j')
         ttime = space(translate(time(),' ',':'),Ø)
         tempds = userid()'.'execname'.D'tdate'.T'ttime
/* Copy the HFS file to an interim MVS dataset                       */
        "OGET '"file"' '"tempds"' TEXT"
         EXITRC = RC
         if EXITRC <> Ø then
            do
```

```
            say 'OGET error' file 'to' tempds 'RC='EXITRC
            exit(EXITRC)
           end
/* Allocate the copy                                                 */
       "ALLOC F(INPUT) DA('"tempds"') SHR"
        EXITRC = RC
        if EXITRC <> Ø then
           do
            say 'ALLOC error on INPUT' tempds 'RC='EXITRC
            exit(EXITRC)
           end
       end
    else
/* Allocate as a DSN                                                 */
       do
       "ALLOC F(INPUT) DA('"file"') SHR"
        EXITRC = RC
        if EXITRC <> Ø then
           do
            say 'ALLOC error on DSN' file 'RC='EXITRC
            exit(EXITRC)
           end
       end
/* Read the DSN                                                      */
   "EXECIO * DISKR INPUT (STEM INPUT. FINIS"
    if RC <> Ø then say 'EXECIO error on' file
   "FREE F(INPUT)"
    if RC <> Ø then say 'FREE error on' file
/* Concatentate the records                                          */
    text = ''
    do i=1 to input.Ø
       text = text || input.i
    end
    message = strip(text)
   end
/* Initialize the API                                                */
 mqrc = RXMQV('INIT')
 if word(mqrc,1) <> Ø then say mqrc
/* Connect to a QMGR                                                 */
 mqrc = word(RXMQV('CONN', qmgr),1)
 if mqrc <> Ø then
    do
     say 'MQCONN to QMGR "'qmgr'" failed MQRC='mqrc
     exit(mqrc)
    end
/* Open the Queue                                                    */
 options = mqoo_inquire+mqoo_output+mqoo_browse+mqoo_set
 mqrc = word(RXMQV('OPEN', qname, options , 'h', 'ood.' ),1)
 if mqrc <> Ø then
    do
     say 'MQOPEN of QUEUE "'qname'" failed MQRC='mqrc
```

```
     exit(mqrc)
    end
/* Format and PUT the message                                          */
 msg.Ø = length(message)
 msg.1 = message
 imd.PER  = MQPER_PERSISTENT
 imd.FORM = MQFMT_STRING
 ipmo.opt = MQPMO_SYNCPOINT
 mqrc = word(RXMQV('PUT', h,'msg.','imd.','omd.','ipmo.','opmo.'),1)
 if mqrc <> Ø then
    do
     say 'MQPUT to QUEUE "'qname'" failed MQRC='mqrc
     exit(mqrc)
    end
 else
    say 'QMGR:' qmgr 'Q Name:' qname 'MSG:' message
/* Inquire on the number of messages on the Q                          */
 atrin = mqia_current_q_depth
 atrou = ''
 mqrc = word(RXMQV('INQ', h, atrin, 'atrou' ),1)
 if mqrc <> Ø then
    do
     say 'MQINQ on QUEUE "'qname'" failed MQRC='mqrc
     exit(mqrc)
    end
 else
    say 'Q Depth:' atrou
/* Close the Q                                                         */
 mqrc = RXMQV('CLOSE', h, mqco_none)
 if word(mqrc,1) <> Ø then say mqrc
/* Disconnect from the QMGR                                            */
 mqrc = RXMQV('DISC', )
 if word(mqrc,1) <> Ø then say mqrc
/* Terminate the API                                                   */
 mqrc = RXMQV('TERM', )
 if word(mqrc,1) <> Ø then say mqrc
```

Robert Zenuk
Systems Programmer (USA)

Creating MQSeries objects with MQAI

The MQSeries Administration Interface (MQAI) is an API that performs administrative functions against MQSeries objects. This article will explore the use of this interface to create a queue. I will

use Visual Basic 6.0 for the sample code but I will try to make the logic clear enough for you to apply it to your own platform.

WHY NOT USE MQSERIES EXPLORER?

A GUI client, such as MQSeries Explorer, is usually the preferred way to create and maintain MQSeries objects. However, there will be times when you may need a higher degree of automation than interactive applications can support. This is when you will want to use MQAI in a custom-built application.

MQAI ARCHITECTURE

MQAI works by sending command messages to a reserved queue named SYSTEM.ADMIN.COMMAND.QUEUE. The queue manager processes the command and sends back a response via a specified response queue. The API takes care of all these details so you just have to tell it what to do.

MQAI organizes data into structures called ‘bags’. You put your request in an administrative bag and the response is contained in a response bag, which contains a system bag for each different piece of response data.

SECURITY REQUIREMENTS

Aside from the obvious authorizations, such as connecting to the queue manager and creating queues, you need specific access to:

- SYSTEM.ADMIN.COMMAND.QUEUE (put).
- SYSTEM.DEFAULT.MODEL.QUEUE (get, inq, dsp).

CREATING THE PROGRAM

You will find it helpful to include three header files in your VB project, which IBM supplies with the MQSeries Client for API support.

- CMQB.BAS – contains constants for defining queue properties.
- CMQBB.BAS – contains API declarations for the MQAI API.
- CMQCFB – contains command constants for the MQAI API.

These modules were originally supplied for VB 4.0 and some of the syntax needs to change for later versions of VB. If you remove the 'Global' keyword on the constant declarations it should work fine.

Step 1

Connect to your queue manager. Either the ActiveX or Win32 API will work. Most error handling is omitted from this sample for the sake of brevity.

```
Dim intCompCode As Long, intReason As Long
Dim intConn As Long
MQCONN "myQMgr", intConn, intCompCode, intReason
```

Step 2

Create your data bags with the *mqCreateBag* API call.

```
Dim adminBag    As Long    ' Admin bag handle.
Dim systemBag   As Long    ' System bag handle.
Dim responseBag As Long    ' Response bag handle.
adminBag = MQHB_UNUSABLE_HBAG
systemBag = MQHB_UNUSABLE_HBAG
responseBag = MQHB_UNUSABLE_HBAG
' Create an admin bag.
mqCreateBag MQCBO_ADMIN_BAG, adminBag, _
            intCompCode, intReason
' Create a response bag.
mqCreateBag MQCBO_ADMIN_BAG, responseBag, _
            intCompCode, intReason
' No need to create the system bag.
```

Step 3

Create your request. The bare minimum for creating a queue is that you supply the queue name and queue type. You add string parameters to the administrative bag with the mqAddString API call. For numeric parameters use mqAddInteger. The valid queue type constants are defined in *cmqb.bas* as: MQQT_LOCAL, MQQT_MODEL, MQQT_ALIAS, MQQT_REMOTE, and

MQQT_CLUSTER.

```
' Put queue name into admin bag.
mqAddString adminBag, MQCA_Q_NAME, _
            MQBL_NULL_TERMINATED, _
            "MILLS.TEST.QUEUE", _
            intCompCode, intReason
' Make the queue type local.
mqAddInteger adminBag, MQIA_Q_TYPE, MQQT_LOCAL, _
 intCompCode, intReason
```

Step 4

Execute your request with the **mqExecute** API command. The constant MQCMD_CREATE_Q tells MQAI to use the parameters in the administrative bag to create a queue. You can find other useful command constants in *cmqcfb.bas* by searching for the 'MQCMD_' prefix. Notice that the handle for the response bag is included with the command arguments. If the command executes successfully it will fill that bag with response information.

```
mqExecute intConn, MQCMD_CREATE_Q, _
          MQHB_NONE, adminBag, responseBag, _
MQHO_NONE, MQHO_NONE, _
intCompCode, intReason
```

Step 5

Evaluate the results. Usually, error checking with MQSeries is a straightforward process. You check the completion code and if it is greater than zero you get the cause of the error from the reason code. It is more complex with MQAI because the reason code usually just tells you nothing more than that the command failed. You need to get the reason for the failure from the response bag as shown below.

The **mqInquireBag** command gives you a handle to the system bag, which you need to access the system bag values you want. The command **mqInquireInteger** returns the data you need into variables intMqExecuteCC and intMqExecuteRC. Now you have the real completion and reason codes.

You can request other values from the response bag where applicable. You can find these selector constants in *cmqbb.bas*. Integer constants have the 'MQIASY_' prefix. Other system bag

selectors are in *cmqb.bas*. Look for the prefixes ‘MQCA_’ and ‘MQIA_’.

```
Dim intMqExecuteCC As Long
Dim intMqExecuteRC As Long
If intCompCode = MQCC_OK Then
    MsgBox "Queue created successfully."
    Exit Sub
ElseIf intReason = MQRCCF_COMMAND_FAILED Then
     mqInquireBag responseBag, MQHA_BAG_HANDLE, Ø, _
systemBag, intCompCode, intReason
     mqInquireInteger systemBag, MQIASY_COMP_CODE, _
MQIND_NONE, intMqExecuteCC, _
intCompCode, intReason
     mqInquireInteger systemBag, MQIASY_REASON, _
MQIND_NONE, intMqExecuteRC, _
               intCompCode, intReason
     MsgBox "Create attempt failed CC: " & _
            intMqExecuteCC & _
            " RC: " & intMqExecuteRC
Else
     MsgBox "Create attempt failed CC: " & _
            intCompCode & " RC: " & intReason
End If
```

CREATING A MORE COMPLEX QUEUE

Back in step 3 I showed you how to specify the queue name and type to MQAI. You can specify additional properties in the same way. Constants representing each property are defined in *cmqb.bas*. The string parameters are prefixed with ‘MQCA_’ and integers with ‘MQIA_’. Use the appropriate **mqAddString/ mqAddInteger** command. Here is an example of adding properties for the backout threshold and backout re-queue name:

```
' Set the backout threshold to 1.
mqAddInteger adminBag, MQIA_BACKOUT_THRESHOLD, 1, _
intCompCode, intReason
' Set the backout requeue name.
mqAddString adminBag, MQCA_BACKOUT_REQ_Q_NAME, _
MQBL_NULL_TERMINATED, "MILLS.TEST.ERROR", _
intCompCode, intReason
```

WORKING WITH OTHER OBJECTS

You can easily apply what we’ve covered in this article to other

MQSeries objects. There are **MQCMD_*** command constants for queues, queue managers, processes, channels, name lists, and clusters. Simply choose the appropriate command constant for the **mqExecute** call, and the right properties for the **mqAddInteger** and **mqAddString** calls.

MQAI COMMAND SYNTAX

The MQAI API calls are documented in Chapter 5 of the *MQSeries Administration Interface Programming Guide and Reference* in your MQSeries manuals. Here is a brief description of the ones used in this article:

mqCreateBag (Options, Bag, CompCode, Reason)

- Options – long: the type of bag to create. The recommended value is MQCBO_ADMIN_BAG.
- Bag – long: this is the handle to the bag you are creating.
- CompCode – long: MQSeries completion code.
- Reason – long: MQSeries reason code.

mqAddString (Bag, Selector, BufferLength, Buffer, CompCode, Reason)

- Bag – long: this is the handle for your administrative bag.
- Selector – long: selects the property you are adding to the bag.
- Bufferlength – long: the length of the string you are adding. Since VB strings are null terminated you can use the constant MQBL_NULL_TERMINATED instead.
- Buffer – string: the text string you are adding to the bag.
- CompCode – long: MQSeries completion code.
- Reason – long: MQSeries reason code.

mqAddInteger (Bag, Selector, ItemValue, CompCode, Reason)

- Bag – long: this is the handle for your administrative bag.
- Selector – long: selects the property you are adding to the bag.
- ItemValue – long: the numeric value you are adding to the bag.
- CompCode – long: MQSeries completion code.
- Reason – long: MQSeries reason code.

mqExecute (Hconn, Command, OptionsBag, AdminBag, ResponseBag, AdminQ, ResponseQ, CompCode, Reason)

- Hconn – long: queue manager connection handle.
- Command – long: an **MQCMD_*** command constant.
- OptionsBag – long: use MQHB_NONE.
- AdminBag – long: your administrative bag handle.
- ResponseBag – long: your response bag handle.
- AdminQ – long: handle to an open command queue. Using MQHO_NONE will default to SYSTEM.ADMIN.COMMAND .QUEUE.
- ResponseQ – long: handle to an open response queue. Using MQHO_NONE will default to a dynamic queue which is automatically deleted on completion of the mqExecute call.
- CompCode – long: MQSeries completion code.
- Reason – long: MQSeries reason code.

mqInquireBag (Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)

- Bag – long: your response bag handle.
- Selector – long: a constant representing the item you want from the bag. For example, MQHA_BAG_HANDLE.
- ItemIndex – long: index of the item you want to access.
- ItemValue – long: handle to the object you are accessing – in

our case, the system bag. This parameter is filled in by the call.

- CompCode – long: MQSeries completion code.
- Reason – long: MQSeries reason code.

mqInquireInteger (Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)

- Bag– long: your system bag handle. This was initialized with an mqInquireBag call.
- Selector – long: a constant representing the item you want from the bag. For example, MQIASY_REASON.
- ItemIndex – long: index of the item you want to access, or MQIND_NONE if there is only one occurrence of the value.
- CompCode – long: MQSeries completion code.
- Reason – long: MQSeries reason code.

mqInquireString (Bag, Selector, ItemIndex, Bufferlength, Buffer, StringLength, CodedCharSetId, CompCode, Reason)

- Bag– long: your system bag handle. This was initialized with an mqInquireBag call.
- Selector – long: a constant representing the item you want from the bag. For example, MQIASY_REASON.
- ItemIndex – long: index of the item you want to access, or MQIND_NONE if there is only one occurrence of the value.
- Bufferlength – long: the length of the string you are adding.
- Buffer – string: a string variable to receive the output from this call. This string should be set to contain as many null characters as you specified in BufferLength.
- StringLength – long: an output parameter indicating the length of the string actually returned.
- CodedCharSetId – long: set it to zero if you don't need to convert your character set.

- CompCode – long: MQSeries completion code.
- Reason – long: MQSeries reason code.

Mills Perry
IT Consultant/Instructor
ZyQuest (USA)

MQ news

Reconda International, providers of Web-based software products that facilitate and enhance WebSphere MQ messaging application development, testing, and support, showed off its newest software solution, QN-StatWatch, at the 2003 IBM Transaction and Messaging Conference in Las Vegas, Nevada, which ran 9-12 February.

Designed to collect statistics at the channel, queue, and message level, QN-StatWatch is claimed to be the industry's first browser-based WebSphere MQ and WMQI support solution that provides MQ administrators, system architects, and managers with the data they need to facilitate accurate charge-back modelling, SLA compliance, and resource capacity planning.

Reconda claims that the product delivers an unprecedented level of security while providing the statistical support, centralized management, and reporting capabilities today's businesses need to analyse and leverage application and business performance on a global scale.

For more information contact:
Reconda, 15 East Putnam Avenue, Suite 306, Greenwich, CT 06830, USA.
Tel: (203) 299 4000.
Fax: (203) 299 4095.
Web: www.reconda.com

* * *

IBM has recently announced the release of CICS Business Event Publisher for MQSeries V1.1, which enables a rapid extension of existing applications running in CICS Transaction Server V1.3 or CICS Transaction Server V2.2.

Business Event Publisher generates user-defined MQSeries messages as a side effect when certain EXEC CICS commands are executed by a CICS application. This message generation is transparent to the application program, so these remain unchanged when Business Event Publisher is used.

Rules control the generation of the MQSeries messages, which are defined using a Microsoft Windows-based graphical utility. These rules enable the content of the MQSeries message to be customizable, as is the queue to receive the message.

Business Event Publisher rules can match, for example, VSAM file updates or temporary storage activity and so notify another application of a change to a record.

For more information contact your local IBM representative.

* * *