# 68

# MQ update

*February 2005*

## In this issue





A itc PUBLICATION

# MQ Update


## Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

## Editor

Trevor Eddolls
E-mail: trevore@xephon.com

## Publisher

Colin Smith
E-mail: info@xephon.com


## 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 $380.00 in the USA and Canada; £255.00 in the UK; £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 2000 issue, are available separately to subscribers for $33.75 (£22.50) each including postage.

## Contributions

When Xephon is given copyright, articles published in *MQ Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please 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.*

# The WebSphere MQ for Unix administration tool

## INTRODUCTION

The utility **mq** has been designed to make an MQ administrator's life much easier and more reliable. It's powerful, and it's freeware! Are you tired of typing a command for every single entry when manipulating a set of MQ objects the same way? Do you think **runmqsc** should allow for wildcards anywhere in an object's name and not only at the end? Are you looking for a cluster agent, safely (and we mean really safely!) controlling your MQ resources? Does MQ for Solaris strain your nerves once in a while, because you cannot clean the shared memory, even though all queue managers have been stopped? Are you longing for a nice short command to inspect a queue manager's error logs that are hidden down an endless directory path? Would you like to start and stop all queue managers on a system with one short command you can type in one second? Would you like **endmqm** to clean up your memory automatically? If you answered any of the above questions with yes, **mq** is the utility for you.

The utility **mq** is a shell program calling, interactively or in batch mode, utilities from WebSphere MQ and the Unix shell. It supports MQ administration from the shell in a much more powerful and comfortable way than the original tools by IBM. The script **mq** is very flexible with regard to error logging, is robust, and provides a suitable return code at a user-defined timeout – regardless of the state of the system resources (eg SCSI timeouts of disks). Hence it can easily and reliably be used as a cluster agent (eg in a Veritas cluster). It is suitable for remote administration, including z/OS queue managers, almost as if the remote queue managers were on the local system.

The Unix derivatives currently supported are AIX, Linux, HP-UX, and Solaris. Along with the script **mq**, both a PDF manual

and man page are available, which contain more thorough technical information than can be supplied by this article.

## SYNOPSIS OF MQ

The utility **mq** carries out the following actions on the resources specified by the subsequent arguments:

- Starts, stops, or kills WebSphere MQ queue managers.
- Displays a qmgr's error log or WMQ processes (ps/pw).
- Pings qmgrs or runs WMQ commands for one qmgr (using **runmqsc**).
- Alters, clears, deletes, or displays WMQ objects.
- Displays the number of active clients per qmgr.

The utility is invoked as follows:

```
mq    action  [ -ailmpstx ]   [ -c[:pattern] ]   [ -e 'command' ]   [ -o
'command' ]   [ -w timeout ]   [ qmgr | pattern … ]
```

or:

```
mq    mqsc  object  [ -e 'command' ]  [ -o 'command' ]  [ -w timeout ]
[ qmgr | pattern … ]
```

where:

- action={ cl | kill | log[2|3] | ps | pw | run | start | stop }.
- command=shell command for logging (see later).
- mqsc={ alt[er] | clear | delete | dis[play] | ping }, WMQ command as for **runmqsc**.
- object=WMQ object specification of the form:
  'type [ (pattern) ]  [ attribute [ (pattern) ]  [ … ]  ]'
  where: *type* denotes any type of MQ object (eg QLOCAL) and *attribute* any attribute associated with this MQ object.
- pattern=pattern to be matched with queue manager names, object names, or attributes; case insensitive full regular expression.

- qmgr=one or more queue manager name(s). Instead of a full qmgr name, a case-insensitive pattern may be specified that identifies the queue manager(s). If *qmgr* is missing, **mq** addresses the local default queue manager, or, if no default queue manager is defined, the first queue manager identified on the system (in */var/mqm/mqs.ini*).
- timeout=timeout in seconds.

See **mq**'s PDF manual or the man page for a full description of all command line options. The *action*s, in alphabetical order, basically do the following:

- **cl** displays the number of active server connections of the *qmgr*(s).
- **kill** pre-emptively terminates all processes of one queue manager.
- **log/log2/log3** opens in current default editor the corresponding error log of the *qmgr.*
- **ping** pings a queue manager using the **mqsc** command **ping qmgr** (**display qmgr** on z/OS, where **ping** is not valid).
- **ps** displays WMQ processes, filtered according to the *pattern*(s).
- **pw** does the same as *ps* but tidily wraps lines longer than 80 characters.
- **run** runs WMQ commands for the *qmgr* (ie calls **runmqsc**). If no local queue manager is found matching the *pattern*, a matching remote queue manager is looked up (and command line options added to **runmqsc** in the background as needed).
- **start/stop** starts or ends one or several *qmgr*(s). If the *qmgr* is omitted, **mq** starts or stops all local queue managers on the system. By default, with each queue manager it starts and stops the following:

- a channel initiator
- a command server
- a trigger monitor
- a listener with a port associated according to the configuration (unless the port is configured for the inet daemon).

After stopping a qmgr, the semaphores and the inter-process communication memory of the queue manager(s) are cleaned up.

All services associated with a queue manager may be controlled individually by various command line options, ie:

- -i – starts or stops the standard channel initiator of the *qmgr*(s).
- -l – starts or stops the default listener configured for the *qmgr*(s) (by **runmqlsr** and **endmqlsr**).
- -m – starts the *qmgr*(s) (by **strmqm**), each with its standard channel initiator (unless CHINIT=NO is specified in the QueueManagerStartup section of the file *qm.ini*), but without other WMQ services unless explicitly flagged by **-i**, **-l**, **-s**, and/or **-t**. With stop (by **endmqm –i**) the *qmgr* takes down all its services except its listener.
- -s – starts or stops the command server of the *qmgr*(s) (by **strmqcsv or endmqcsv**).
- -t – starts or stops the standard trigger monitor of the *qmgr*(s) (by **runmqtrm** and **kill**).

## REGULAR EXPRESSIONS WITH MQ

Regular expressions (regex) are a very powerful tool for matching text patterns. Only a brief, **mq**-specific tutorial is given in what follows. Regular expressions are documented thoroughly by the **regexp(5)** man pages. In the Unix shell they are used, for example, by the well-known tools **grep**, **sed**, and

**awk**. The entire matching of patterns in **mq** is accomplished by the aid of the latter.

The simplest regex is simply a word or, more generally, a string of characters. A regex consisting of a word matches any string containing that word. The pattern matching in **mq** generally is case insensitive, excluding the *actions ps/pw*.

Not all characters can be used 'as is' in a match. Some characters, called meta-characters, are reserved for use in regex notation. The meta-characters are:

```
[ ] ( ) { } ^ $ . | * + ? \ /
```

Fortunately, most of these characters usually are not present in the names or attributes matched by **mq**. The meta-characters most relevant for matching names and attributes by **mq** are:

- The period (full stop) '.' matches any character.
- The asterisk '*' allows us to determine repeats of a character. It is put immediately after the character (or character class) that we want to specify. With **mq** it is most often used in combination with a period '.*' to specify 'any number of characters of any kind'.
- By putting a backslash '\' before a meta-character, its special meaning is suppressed, eg, '\.' matches a period (full stop).
- The caret '^' is an anchor, matching the beginning of a string, ie '^cat' matches 'cat' or 'catapult' but not 'bobcat'.
- The dollar sign '$' is an anchor, matching the end of a string, ie 'cat$' matches 'cat' or 'bobcat' but not 'cats'.
- The pipe '|' is the alternation character, used in the sense of an 'or', ie 'cat|dog' matches either 'cat' or 'dog'.
- Brackets [...] denote a character class, which allows a set of possible characters to match at a particular point in a regex. Here are some examples:
  - '[bcr]at' matches 'bat', 'cat', or 'rat'.

– ‘[cab][12]’ matches the first occurrence of ‘a’, ‘b’, or ‘c’, followed by ‘1’ or ‘2’, ie ‘a1’, ‘a2’, ‘b1’… (the order of occurrence in the class is not relevant).

The caret ‘^’ in the first position of a character class denotes a negated character class which matches any character but those in the brackets.

– ‘[^a]at’ doesn’t match ‘aat’, but matches all other – ‘bat’, ‘cat, ‘0at’, ‘%at’, etc.

– ‘[^0-9]’ matches a non-numeric character.

## THE NEW GENERATION OF MQ ADMINISTRATION

The utility **mq** provides an interface for command line administration that is much more powerful than **runmqsc** shipped with the WebSphere MQ. Suppose, eg:

- You know the name of a remote queue but not the name of the local remote queue definition pointing to it. With **mq**, you can easily display it.
- You need to find out which queue triggers the mainframe transaction ‘T098’. If the mainframe is connected to your Unix server, **mq** can help. It reveals the matching process definition, from which you find the name of the queue.
- You see that your filesystem is running full, but you don’t know which of your 647 local queues pile up thousands of large messages **mq** can tell you.

When called with an **mqsc** command, **mq** runs the command (by **runmqsc**) on the *qmgr* supplied or, if omitted, on the local default queue manager of the system, or, if no default queue manager is configured, on the first queue manager identified on the system (in */var/mqm/mqs.ini*). If no local queue manager is found matching the *pattern* and a local default queue manager is defined, all available remote queue managers are compared with the *pattern*. Hence remote administration can be performed as if the remote queue managers were local.

When called with an **mqsc** command, **mq** affects all WMQ objects of type *type* matching the *pattern* in the *object* specification. If '(*pattern*)' is omitted completely from *object*, all *object*s of type *type* are affected. For manipulative or destructive commands (ie *mqsc* = alter, clear, delete), the user is prompted for confirmation before the commands are executed.

In order to address a subset of the objects matching the name pattern, a similar pattern may be specified with any attribute of the object. All objects matching both the name pattern (if specified) and the attribute pattern will be selected. A few examples will shed light on the usage of **mq**:

- Display the trigger setting and the current depth of all local queues beginning with 'COMPLAINT' and ending with 'IN', hosted by the remote queue manager OSIRIS:

  ```
  mq dis 'ql(^complaint.*in$) trigger curdepth' osi
  ```

  Note the incomplete queue manager name (requires that no other local or remote queue manager name matches 'OSI' and appears prior to OSIRIS in the WebSphere MQ configuration).

- Display all remote queue definitions on the local default queue manager that contain 'PUBLISHER' in their remote queue manager attribute:

  ```
  mq  dis  'qr  rqmname(publisher)'
  ```

- Set the default persistence of all queues containing 'ORDER' anywhere in their name, hosted by the local default queue manager:

  ```
  mq alt 'q(order) defpsist(yes)'
  ```

  You will be prompted to confirm the whole set of commands generated and, unless the response 'all' is given, for the confirmation of each individual command as well.

- Delete all channels containing either ISIS or OSIRIS anywhere in their name, hosted by the queue manager BROKER1:

```
mq delete 'chl(isis|osiris)' broker1
```

As in the previous example, you will be prompted to confirm the execution.

- Display the current status of all channels of the local queue manager PUBLISHER:

```
mq dis chs pub
```

Note the incomplete queue manager name (requires that no other local queue manager name matches ‘PUB’ and appears prior to PUBLISHER in the WebSphere MQ configuration).

- Display the current depth and maximum depth of all local queues on the local default queue manager that currently contain at least 1000 messages:

```
mq  dis  'ql  curdepth(....) maxdepth'
```

The four dots here denote a number with at least four digits (for curdepth only digits are suitable), ie curdepth ≥ 1000.

## MAKE YOUR MQ ENVIRONMENT SAFE AND STABLE

### Flexible configuration

A configuration section with enterprise-wide defaults for the utility **mq** is found at the top of the script, after the leading comments. Useful defaults are supplied by the author, but the users may change these defaults as they wish. System-specific configurations may be supplied in an optional separate file, *mq.conf*, which may be local to the script **mq** or at */var/mqm*. The local exceptions in *mq.conf* override the standard rules of the internal configuration section.

The following resources can be configured:

- TCP/IP ports for listeners – in order to identify rules for assigning queue manager names to TCP/IP ports that differ from MQ’s default 1414, a list of patterns may be configured.

- Remote administration of z/OS queue managers – in order to enable the remote administration of z/OS queue managers in your environment, a list of name patterns of z/OS queue managers is supplied.
- To prevent the accidental modification of system objects and optionally other hidden objects (specified by the user), patterns are configured, identifying MQ objects not to be affected (unless forced on the command line by flag **-a**).
- Various defaults are specified, eg regarding logging, timeouts, security, or the start-up and shutdown of services with queue managers.

### Start-up automation

In order to have all queue managers regularly started and stopped at system start-up and shutdown, the user may hard link or copy the script **mq** to the directory *etc/init.d* and create the hard links *etc/rc2.d/Snnmq and etc/rc0.d/Knnmq* where *etc* denotes */etc* or */etc/rc.d* (for Linux). The ownership root:mqm and the file mode u:rwx g:rwx o: are recommended. S*nn*mq and K*nn*mq are called by the operating system with a parameter start and stop, respectively starting or stopping with standard options (may be changed by the user) all queue managers on the system. Ordinals *nn* > 80 are recommended. Alternatively, you may add an entry to */etc/inittab*:

```
mqs:2:once:/usr/local/bin/mq start >/dev/console 2>&1
```

(Note: the script **mq** switches to the user mqm; **su – mqm** is not required.)

### Monitoring with mq

Monitoring tools such as IBM Tivoli typically provide a command line interface for posting a message to the monitoring central console. **Mq** is pre-configured to send, when called in batch mode, its output and its error messages to the system logger. However, these logging commands may easily be changed by the user.

In order to enhance the flexibility of the use of such posting interfaces, the options **-e** (command for logging errors) and **-o** (command for logging regular output) have been added to **mq**. By these options the logging of **mq** may be changed from the command line at run-time. The command:

```
mq  start  -e 'postemsg -m $msg -r CRITICAL mq_error MQM'  -mt
```

would start all queue mangers on a system, each with a trigger monitor (without a command server but with a channel initiator unless CHINIT=NO is specified in the QueueManagerStartup section of the file *qm.ini*). The standard output still is directed to the system logger (or to the terminal, if called interactively) but any error messages would be posted to Tivoli as a 'critical event'. Note the variable $msg used in the command option **-e**, which is replaced at run-time with any line of text that (without option **-e**) would be put to the 'error out' (>&2).

### Using mq as a cluster agent

Running MQ in a clustered system is quite a delicate task because MQ is I/O bound. If you are operating MQ in a cluster environment, you will find it very difficult to find a cluster agent that is well-prepared for disk errors. Strangely enough, to the knowledge of the author, neither IBM nor the sellers of cluster software (such as Veritas or HP) or any third-party software vendor has yet come up with a cluster agent safely controlling MQ resources. However, you will need a monitoring, a stop, and especially a cleaning utility for your MQ services that will return at a well-defined timeout, regardless of the state of your system resources. If, for example, the external disk volume hosting your */var/mqm* filesystem should hang, neither **runmqsc** nor **endmqm** will ever return. Hence, without manual intervention, your cluster will fail to switch from one node to the other.

The agent **mq** calls any IBM utility as a controlled sub-process, which is pre-emptively terminated at a user-defined timeout if it did not return. Furthermore, **mq** is very flexible regarding the logging of its output.

IBM recommends cleaning the memory after stopping all your MQ services in order to prevent your system's memory from gradually being used up by semaphores and memory segments that are left behind. If you operate MQ in a cluster based on Solaris, you will probably find that cleaning the memory fails, even after stopping all MQ services on the system.

Your remote queue managers connected to the cluster do not know by themselves that MQ on the clustered system is shutting down. Their sender channel agents keep polling and hence the local inet daemon keeps launching receiver channel agents, which immediately shut back down when they find that their queue manager is not available. The latter prevent the IBM utility **amqiclen** from cleaning the memory and the receiver channel agents themselves leave semaphores and shared memory segments behind.

The cluster agent **mq** is capable of safely deactivating and re-activating the MQ services in the inet daemon's configuration – thus truly and completely shutting down MQ, enabling a smooth clean up of MQ's memory resources.

Operating a Veritas cluster, you may use for MQ a simple application agent with, for example, the following command line to stop a queue manager named PUBLISHER (exact name, with no other leading or trailing characters: flag -x) with a timeout of 90 seconds (option -w), disabling MQ in the inet daemon (flag -p), and logging regular output (option -o) as 'information' entries (Veritas tag E) and errors (option -e) as 'critical errors' (Veritas tag B):

```
mq stop -e 'halog -add B $msg' -o 'halog -add E $msg' -w90 -pmx
PUBLISHER
```

Very similarly, you may configure the start and the clean (by *action* kill) of the queue manager or the start, stop, and clean of any other MQ service, eg the command server, a trigger monitor, or even a listener.

In fact, the consistent configuration of an MQ listener resource in a cluster is rather tricky because, normally, a listener can be started only *after* the queue manger has started but it *refuses*

to stop *before* the queue manager has stopped. Consequently, normal resource dependencies will not work with an MQ listener. With **mq** as a cluster agent, you are free to stop a listener even if the queue manager is still running. Hence you may simply configure the listener as a resource, depending on its queue manager.

## DISTRIBUTION AND SUPPORT

The program **mq** is freeware. Its copyright is owned by the author but it's distributed under the Gnu Public License and hence may be used or even changed free of charge by anyone. The utility is available as the IBM Cat. 4 (third party) SupportPac named ma6o. It is shipped with the following files:

- **mq** – the shell script for the administration of WebSphere MQ services.
- *mq.conf* – optional configuration file for **mq**.
- *mq.1* – manual entry (man page), to be copied, eg, to the directory */usr/local/man/man1*.
- *mq.pdf* – printable manual for **mq**.
- **mqTrc** – shell script for tracing and support; executed (instead of **mq**) in order to generate trace output of **mq** to be mailed to the author for efficient support.
- *gpl-v2.pdf* – Gnu Public License, Version 2.
- *readme.txt* – introduction, quick installation guide, and history of changes.

A zip file containing all these files can be downloaded from www.xephon.com/extras/mqboehm.zip.

Postcards, suggestions, or condolences for debugging about 2000 lines of Bourne shell script code are welcome. You are invited to contact the author if you are using **mq.** Please report bugs and any comments to j.boehm@gmx.ch.

*Dr sc nat Johannes P Boehm*
*IBM certified MQSeries Specialist,*
*IBM certified MQSeries Developer,*
*IBM certified MQSeries Solutions Expert (Switzerland)*

# MQ transactions

## INTRODUCTION

This article discusses the use of distributed transactions for processing MQSeries messages in .NET applications. The sources used in this article were written in Microsoft Visual Studio .NET 2003 in C#. The whole solution can be downloaded from www.nesterovsky-bros.com/downloads/MQTransactions.zip.

## TRANSACTION PROCESSING FUNDAMENTALS

Different computing technologies operate using transaction concepts. There are differences in treatment, but the main meaning is always the same: to execute a series of logically-related operations as one unit of work. Such execution supposes that either all or no operations will complete. Regardless of the technology used to implement transactions, they are using the following basic concepts:

- A transaction has a beginning and an end that specify its scope, which determines a series of operations that are contained within this transaction.
- All operations in a transaction scope must complete successfully in order to make the transaction successful. When a transaction completes successfully, it is committed, which causes all the operations to become permanent and the transaction to end.
- If at least one operation from the transaction scope fails, then all operations will be aborted and rollback occurs, undoing any changes that have already been executed before the abort.

Transactions can also span multiple data resources. Distributed transactions give you the latitude to incorporate several

distinct operations occurring on different systems into a single pass or fail action.

## THE MQ TRANSACTIONS

WebSphere MQ (in future called MQ for brevity) supports transactions in three different ways:

1 An MQ queue manager acts as the unit of work coordinator, and only MQ resources participate in the transactions. The queue manager ensures that all changes to resources in the unit of work are committed or backed out together.

2 An MQ queue manager is the resource manager participating, and acts as the unit of work coordinator. Other resources, such as databases and so on, are also involved in the transactions. The transactions' flows are managed by MQI verbs manually from applications.

3 An MQ queue manager is a participant, but does not act as the unit of work coordinator. Instead, there is an external unit of work coordinator with whom MQ cooperates. In this case each put and/or get operation with the corresponding settings can participate in an automatic transaction.

The first and second ways are very useful, but they are limited to dealing with MQ activities only (the first case) or with manual transaction management (the second one). The third case allows using two-phase commit transactions supported by an external transaction coordinator. This kind of transaction is used in distributed systems. The Microsoft Distributed Transaction Coordinator (DTC) is one such external transaction coordinator. Using DTC allows you to initiate a transaction and then process resources from multiple sources, such as MS SQL Server, MQ, etc, and then commit or roll back all changes, depending on the state of your application.

Note: only the MQ server can participate in distributed transactions managed by DTC. Neither MQ client nor the MQ

extended transaction client can do this. Here is a quotation from a PTF memo for IBM WebSphere MQ for Windows Version 5.3 and IBM WebSphere MQ Express for Windows Version 5.3 Service Level 5.3.0.7 (Fix Pack CSD07) [1]:

> 'Unfortunately at this point in time, there are some unresolved issues using Microsoft Transaction Server as the transaction manager to coordinate the IBM WebSphere MQ Extended Transactional Client, therefore this configuration is not supported at present. For more details and any updates to the supported position, please see APAR number II13466.'

## USING DTC TRANSACTIONS WITHIN .NET FRAMEWORK APPLICATIONS

To use DTC transactions from your .NET applications, you need to implement a class that inherits from the System.EnterpriseServices.ServicedComponent one and then mark it with attributes indicating that it requires a transaction. The declarative transaction attribute specifies how an object participates in a transaction, and is configured programmatically. Within this class you can execute whatever operations under the resources that will be participating in your transaction – this might be sending and/or receiving MQ messages, accessing a database, etc. There is no special coding required indicating that you are participating in a transaction; just the fact that you are calling these resources from within your class instance is sufficient to enlist them into the existing transaction or to start new one. Another attribute tells your class that it should automatically commit or abort the transaction based on what happens inside your code. If the method that performs transactional processing ends without raising any exception, the transaction will be automatically committed, otherwise it will be automatically aborted – if an exception is raised and not handled within the procedure itself. But keep in mind that a physical transaction starts only when a class instance accesses data resources, such as a database, message queue, etc. The entire physical transaction occurs automatically.

There is another useful feature that allows voting in automatic transactions. You can use System.EnterpriseServices.ContextUtil class in your code to control the transaction in a very simple manner. This class has SetComplete and SetAbort methods that vote to commit or abort an outgoing transaction respectively, but keep in mind that the transaction is neither committed nor aborted until the root object of the transaction deactivates. Further, a single abort vote from any object participating in the transaction causes the entire transaction to fail. So, the use of voting leverages automatic transactions greatly.

Take a look at the *.NET Framework Developers Guide* documentation[2] for more details of creating, registering, and using serviced components and for details of voting in transactions[3].

## THE SAMPLE

Now it's time to look at the sample referred to in this article. This is an application that retrieves a message from MQ and inserts it into the MS SQL database in one unit of work. There were at least two crucial elements in the development of this application:

- The MQ listener that detects new messages.
- The method that performs the transactional processing on the message once it has been detected.

The listener will browse an input queue, and whenever it detects a new input message, it does the following:

- Passes control to the method that starts the DTC transaction.
- Retrieves the message from the queue in the transaction context.
- Notifies a user about the received message.

Such an approach allows us to avoid the issues of a dummy

transaction. Otherwise, the dummy transaction will lead to excess consumption of system resources.

The MQ listener's implementation is in the *MQListener.cs* file. The main loop that watches the input queue is implemented in the Listen() method, which is executed in a background thread:

```
/// <summary>
/// The main worker loop.
/// </summary>
private void Listen()
{
  if (OnTrace != null)
    OnTrace("MQ listener started.");

  try
  {
    MQQueueManager queueManager = null;
    MQQueue queue = null;
    DTCWrapper transaction = new DTCWrapper();
    DTCMethod receive = new DTCMethod(Receive);

    try
    {
      needRescan = false;
      bool first = true;
      int reason = Ø;

      while(true)
      {
        try
        {
          if (queueManager == null || !queueManager.IsConnected)
          {
            OpenQueue(out queueManager, out queue);
          }

          if (!Browse(first, queue))
          {
            if (needRescan)
            {
              needRescan = false;
              first = true;
            }
            continue;
          }
        }
        catch (MQException e)
```

```
            {
              CloseQueue(ref queueManager, ref queue);
              first = true;
              needRescan = false;

              // Notify about error
              if (reason != e.Reason)
              {
                reason = e.Reason;

                if (OnError != null)
                  OnError(e);
              }

              continue;
            }

            reason = 0;
            first = false;

            // This is a crucial point in MQListener:
            // at this point the MQ message is retrieved from
            // input queue in context of DTC transaction.
            // Note: the Receive() method generates OnReceive
            // event, so user can implement its own processing
            // during the same DTC transaction.
            transaction.Process(receive);
          }
        }
        finally
        {
          CloseQueue(ref queueManager, ref queue);
        }
      }
      catch
      {
        // Thread is stoping...
      }
      finally
      {
        if (OnTrace != null)
          OnTrace("MQ listener stopped.");
      }
    }
```

In general this method does the following:

1 Creates a transactional object (an instance of DTCWrapper class) that is inherited from System.EnterpriseServices.ServicedComponent.

2 Starts an endless loop. Actually the loop can be broken only from a foreground thread by the Thread.Abort() method.

3 Checks the connection to MQ server and reopens it, if needed.

4 Browses the input queue in order to detect a new incoming message.

5 If the queue is empty the loop continues browsing (Step 4).

6 Whenever the listener catches an MQ exception, it closes the current connection, and notifies the user about the error. Then the browsing loop is restarted (Step 3).

7 If there is no error, the Receive() method is called in the transactional context. So, an automatic DTC transaction starts. The Receive method generates an OnReceive event to notify the user of a retrieved MQ message and allows processing of it in the context of the DTC transaction.

8 Independently of whether the transaction is committed or rolled back the loop continues from Step 3.

Note: ensure that the MQ connection is created inside the Listen() method. Although MQSeries supports handle sharing between threads, it is possible only in blocking mode. This means that a connection handle allocated by one thread of a process can be used by other threads belonging to the same process. However, only one thread at a time can use any particular handle. If a thread tries to use a handle that is already in use by another thread, the call blocks (waits) until the handle becomes available.

Note: transactions will be executed sequentially, one by one, when the next message is detected in an MQ queue. It's quite easy to modify the Listen() method so that it will start the Receive() method asynchronously, but in this case the messages from the queue will be processed in an unpredictable order.

The second important method for the MQ listener is Browse():

```
/// <param name="queue">determines the queue to be browsed.</param>
/// <returns>true whenever the new message was found.</returns>
private bool Browse(bool first, MQQueue queue)
{
  MQGetMessageOptions options = new MQGetMessageOptions();
  options.MatchOptions = MQC.MQMO_NONE;
 options.Options = MQC.MQGMO_WAIT | MQC.MQGMO_ACCEPT_TRUNCATED_MSG |
    (first ? MQC.MQGMO_BROWSE_FIRST : MQC.MQGMO_BROWSE_NEXT);
  options.WaitInterval = DefaultTimeout;

  try
  {
    queue.Get(msg, options, 1);

    return true;
  }
  catch(MQException e)
  {
    switch (e.Reason)
    {
      case MQC.MQRC_NO_MSG_AVAILABLE:
        return false;
      case MQC.MQRC_TRUNCATED_MSG_ACCEPTED:
        return true;
      default:
        throw;
    }
  }
}
```

It browses the specified queue. It makes an attempt to detect a new message. The listener doesn't retrieve the whole message when it is detected, but, rather, it returns true to the caller, which means that the transactional processing of the detected message can be started. Otherwise it returns false.

The next method to be considered is Receive():

```
private void Receive()
{
  MQQueueManager queueManager = null;
  MQQueue queue = null;

  try
  {
    OpenQueue(out queueManager, out queue);
```

```
        MQGetMessageOptions options = new MQGetMessageOptions();
        options.Options = MQC.MQGMO_NO_WAIT | MQC.MQGMO_SYNCPOINT;
        options.MatchOptions = MQC.MQMO_MATCH_MSG_ID;

       // a DTC transaction will start only if get is successful
        queue.Get(msg, options);

        if (OnReceive != null)
          OnReceive(msg);

        if (OnTrace != null)
          OnTrace("Transaction is about committed.");
      }
      catch(Exception e)
      {
        if (!(e is MQException))
        {
          if (OnError != null)
            OnError(e);
        }

        Rescan();

        // vote to rollback transaction
        ContextUtil.SetAbort();

        if (OnTrace != null)
          OnTrace("Transaction is rolled back.");
      }
      finally
      {
        CloseQueue(ref queueManager, ref queue);
      }
    }
```

It's a rather short and straightforward method, which, although simple, is crucial, and is performed in a transactional context. It opens a new MQ connection, gets the message under the cursor, and generates an OnReceive event. If any exception is thrown, the method generates an OnError event and votes to roll back the whole transaction. Otherwise the transaction is considered committed automatically. The MQ connection is closed at the end of the Receive() method.

Make sure that the retrieval of the message from the queue is performed using the MQC.MQGMO_SYNCPOINT option. This

notifies the MQ manager that it is participating in a distributed transaction.

To perform common transactional processing, use the DTCWrapper class inherited from System.EnterpriseServices.ServicedComponent:

```
namespace NesterovskyBros.ServicedComponents
{
  using System;
  using System.EnterpriseServices;

  /// <summary>
  /// Defines method to execute in a DTS context.
  /// </summary>
  public delegate void DTCMethod();

  /// <summary>Defines a transactional object.</summary>
  [Transaction(TransactionOption.RequiresNew, Timeout=60)]
  public class DTCWrapper: ServicedComponent
  {
    /// <summary>Executes method within transaction.</summary>
    /// <param name="method">Method to execute.</param>
    [AutoComplete]
    public void Process(DTCMethod method) { method(); }
  }
}
```

This class defines the Process() method only, which is executed in the context of an automatic DTC transaction. The DTCMethod is a delegate (a reference type that can be used to encapsulate a method with a specific signature) that allows the passing of any suitable external method into DTCWrapper.Process() method. Whenever this method is called, a new context for the DTC transaction will be created. The default timeout for a transaction is set to 60 seconds.

Note: some properties of an instance of ServicedComponent can be tuned through the Component Services console.

To check this sample you also need to create a table with the following structure in a MS SQL database:

```
Create table records (
      Id timestamp primary key,
```

```
        rawData ntext
);

Go;
```

Now you can run the sample.

## CONCLUSION

The article has discussed only one way of using MQSeries with DTC transactions within .NET applications. It provides a reliable and easy way to deliver messages from different applications on discrete platforms to .NET applications on a Windows platform.

A zip file containing sample files and transactions can be downloaded from www.xephon.com/extras/mqsrc.zip.

## REFERENCES


1 PTF Memo for: IBM WebSphere MQ for Windows Version 5.3 and IBM WebSphere MQ Express for Windows Version 5.3 Service Level 5.3.0.7 (Fix Pack CSD07) – www-306.ibm.com/software/integration/mqfamily/support/memos/win/memo.txt.

2 MSDN, *.NET Framework Developers Guide*, *Processing transactions* – msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconautomatictransactionsnetframeworkclasses.asp.

3 MSDN, *.NET Framework Developers Guide*, *Voting in automatic transactions* – msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconvotinginautomatictransaction.asp.



*Arthur Nesterovsky*
*Software Developer (Israel)*

# Making better use of MMC for WMQ and WBIMB on Windows

Anyone who uses WMQ Explorer on a Windows platform will be aware that the display is based on Microsoft Management Console (MMC). If you need to access the WMQ services information (to check the listener or channel initiator etc), another MMC session needs to be opened.

MMC allows you to add snap-ins to an existing display of any other administration system using MMC. These include not only the WMQ Services information but also the Windows Services and the Windows Event Viewer. So it's possible to have all four of these administration facilities available under one MMC session. These facilities are constantly needed by anyone developing and testing WBIMB message flows, so to have them all together can be very efficient. This MMC

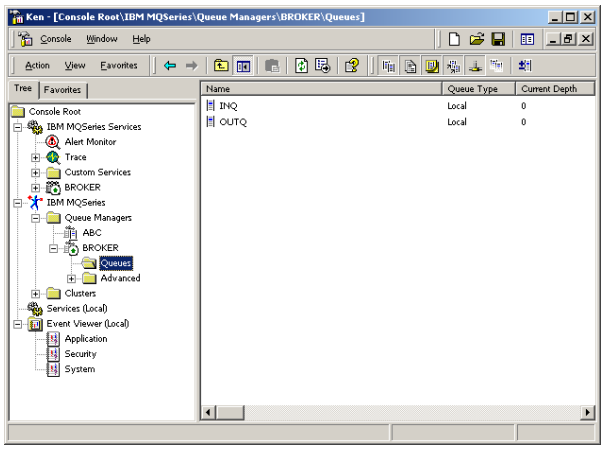


*Figure 1: All chosen sessions displayed*

session can be saved as a file with the .msc extension and then invoked via the task bar, or desktop icon, and it will immediately show all the facilities in one window.

To add a snap-in to your WMQ Explorer session, select **console** from the toolbar and choose **add/remove snap-in**. The resulting window will show the snap-ins currently displayed. Click the **Add** button to reveal a list of further MMC sessions available. Select the desired snap-in from the list and click **Add**. When you return to the display, all the chosen sessions will be available under the single WMQ Explorer window – see Figure 1.

*Ken Marshall*
*Middleware Consultant*
*MQSolutions (UK) Limited*


## Federated identity overview: identities without boundaries

ABSTRACT

How confident are consumers and businesses in the use of the Internet for transactions other than the narrow scope that exists today? The power of the technology components that enable B to B and B to C commerce, and the pervasive nature of the Web, points to the fact that the Web is underutilized as a transaction environment. The notion of federated identities is poised to expand the use of Web services and create the next frontier for identity and access management. This is the first of a two-part series in a study on federated identity management.

Federated identity management makes it possible for someone who has been authenticated (an authenticated identity) in one security domain to access personalized services in other security domains, using Web services. As organizations realize the business benefits of real-time networked processes, this business model is maturing. Deployment of a federated identity infrastructure involves controlling access rights to applications in

a loosely-coupled distributed environment, with a single authentication being the foundation for this access. Utilizing resources on disparate systems that may traverse security domains makes for some challenging technology and business concerns.

The major technology challenges are interoperability, security, and data integrity, while the major business challenges revolve around trusting that identities have been properly authenticated and maintaining regulatory compliance in the new networked environment. Standards bodies were therefore used to define the rules for the development of the interoperability, security, and information integrity components of this new business model. Dealing with the business challenges, however, has proven to be more complex, making the legal components of federated identity management agreements important. Part one of this document will focus on the concepts of federated identity and discuss the standard bodies that are driving the adoption of this new business model. Part two will focus on business drivers, challenges, and give deployment examples.

## INTRODUCTION

Organizations continue to expand the utilization of their systems beyond their internal users, to reach out to their customers, suppliers, and business partners to realize the full potential of the Web. This means allowing end users both within and outside the organization to access and share data and services, regardless of where the users reside. Federated identity management makes it possible for someone who has been authenticated (an authenticated identity) in one security domain to access personalized services in other security domains, using Web services. The fundamental problem that needs to be solved to accommodate these end users is integration or interoperability of the disparate systems and applications. Over time several approaches have evolved, including middleware and Web services. These different technologies solve different aspects of the problem for organizations. Here are some of the questions that govern the type of approach required:

- Is data exchange from one type of software application to the other required?
- Is service level integration a necessity?
- Are the users inside or outside the company?
- Who are the users?
- Is expansion of services to users outside of the defined enterprise directory required to achieve the business goals?
- Is the application distributed?

For application connectivity and interoperability in a heterogeneous environment, middleware technology is used. However, these middleware services employ proprietary implementations and create a high level of difficulty for developers in their coding and management.

With the advent of Web services, proprietary implementation challenges are negated and the coding and management difficulty of services is reduced.

## TECHNOLOGY FOR INTEGRATION

### Middleware technology

The communication between applications and programs that reside on different platforms can be accomplished by connectivity middleware. This connectivity software evolved to support the move to client/server architecture and the development of distributed applications. Initiatives in this area are the Open Software Foundation's Distributed Computing Environment (DCE), Microsoft's Component Object Model (COM) and Distributed Component Object Model (DCOM), and the Object Management Group's Common Object Request Broker (CORBA). These technologies provide a set of APIs for the operating systems and network services that allowed an application to locate services across the network. These technologies are collectively referred to as Enterprise

Application Integration technology and take the following forms.

Transaction processing monitors, eg XCPI-C for request/reply transactions in a distributed environment, provide the tools for developing and deploying distributed applications. For distributed system services, Remote Procedure Calls (RPC) enable application logic to be distributed across the network, so that program logic on remote systems can be executed, by calling a local routine. Message Oriented Middleware (MOM) is used for data exchange between programs, creating the environment for distributed applications. In the MOM environment, message brokers are used to transform the message formats. For example, an application program on an NT machine can request inventory data from a SAP R/3 z/OS application. MOM middleware and message brokers translate the format of the NT message so that it can be understood by the z/OS application, and then reformat the z/OS response for assimilation by the NT application.

Middleware product implementations are unique to the vendor and these proprietary technologies and APIs create vendor lock in. For example the API developed for access to SAP R3 cannot be used to access another ERP application, and that API is proprietary to the vendor. In addition, design choices and functionality make coding difficult. For example, the developer must know what services he must code on the client side and what services to code for the server side in a client/server environment. In addition the DCE and CORBA support unique platform implementations. This type of integration is referred to as ‘tightly coupled’ because it is dependent on architecture and implementations for integration.

### Web services technology

Because of the continued desire of organizations to link customer applications and provide Web-based services, a new generation of enabling technology has emerged that exploits the ubiquity of the Web and more easily supports a

Services-Oriented Architecture (SOA). This technology uses object-oriented technology to wrap data and program elements, and unlike the middleware technologies, provides a standardized way of dealing with integration. The standards employed for this service level integration are Universal Description, Discovery and Integration (UDDI), Web Services Description Language (WSDL), and Simple Object Access Protocol (SOAP). They provide the common way for program-to-program communication. The standards employed ensure that the same communication language is used among constituents. For example, communication between two English speaking individuals is (often) simple, since they have standardized on the language for the exchange of information. However, communication between an English and a German individual, each insisting on using their own language, cannot be understood by either party and an interpreter (transaction broker equivalent) will have to be used. Standards remove the dependency on architecture or environments for successful communication, and do not create vendor lock in. This model of integration is referred to as 'loosely coupled'. In these environments the focus is on enabling services to users outside the trusted defined enterprise directory. It follows therefore, that strong authentication is required because it is this authentication that determines the access privileges. Unlike EAI technology, Web services was specifically designed to be used in a distributed environment.

## BUSINESS CHALLENGES OF THE FEDERATED ENVIRONMENT

Because users outside of the enterprise directory are allowed to access services, many business challenges arise. Businesses should ask: should I allow users from outside of my directory into my environment? Who are these users? How can I trust them in my environment? How should they be authenticated? Can I trust the source or originating domain? What services should I authorize these users to utilize? Can I trust that they are authenticated? Would any regulatory compliance be violated when these users are in my

environment? Are there auditing capabilities that can allow me to monitor everything that is done in my environment?

What safeguards should I put in place for indemnification if my trust is broken? What type of business relationship must exist between my organization and the external organization? How is this relationship different from my already established Web page access? What is the benefit to the organizations involved?

In a federated environment, organizations agree to allow users, after authenticating in their home domain, to access personalized services in other domains. This means that participants in this networked community must trust each other to authenticate their users. A Federated identity is therefore an identity that, once it has been authenticated at initial sign on in the home domain (the sign on authority), can traverse several domains, to perform multiple business processes, without re-authenticating, . The sign-on authority

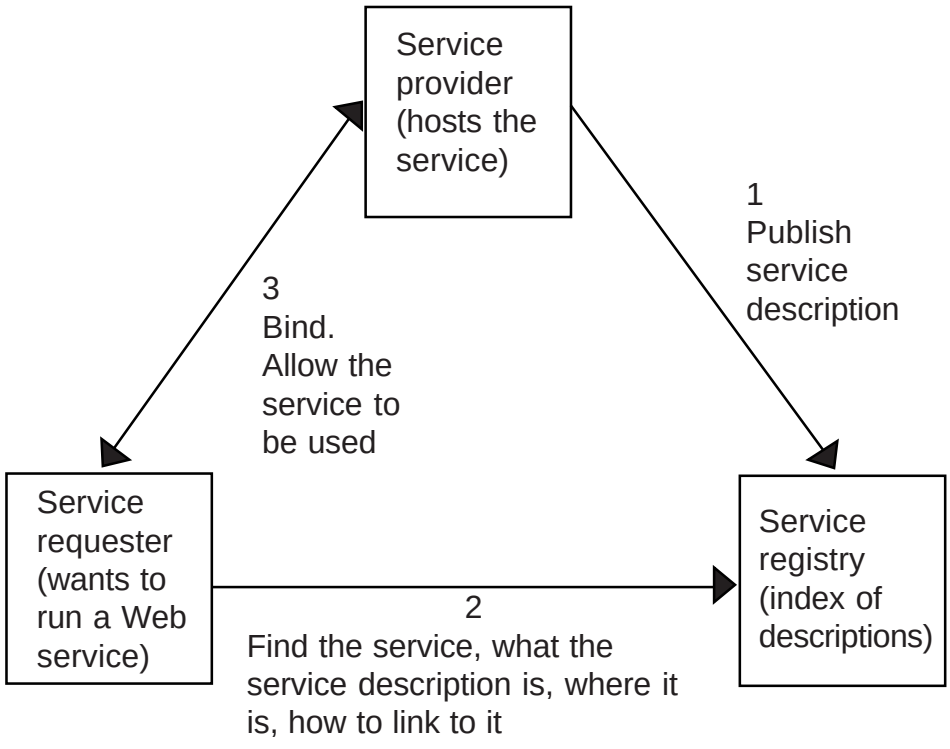


*Figure 1: Web services architecture*

is responsible for the initial authentication, and the other sites in the federated community trust that the login profile is valid to obtain the services requested. The various sites where the applications or services reside are responsible for authorization or granting permissions based on the login profile.

This federation or group of organizations that allow this cross domain access to their applications have been referred to as belonging to a 'circle of trust', since these domains must trust the identities that enter and access their applications. This 'circle of trust', or Web-based collaborative environment, is used to form strategic partnerships, extending business relations between these organizations, creating a 'virtual enterprise'. Federated identity environments employ simplified sign-on or single sign-on, and single log-out, for efficiency and enhanced security.

## CONCEPTS

A study of federated identities involves the discussion of several concepts.

Single sign-on allows a user to sign-on once at the authoritative portal interface, and, after being authenticated, the identity can seamlessly traverse other security domains (or get access to other applications within the domain) without further sign-on. There is also no requirement for the user's personal information to be stored centrally.

In the federated ID environment, single log-out is also an important concern and a necessary function. Single Log-Out (SLO) is the synchronized termination across all sessions in the application domains where the ID was authenticated and authorized to access an application.

Network identity is the integration of the identity security of the enterprise infrastructure (strong authentication, user provisioning, SSO technology, and Web services delivery) with the transport layer (Transmission Control Protocol – TCP) security of the network.

Web services allow applications to offer common interfaces over an HTTP network, ie the Internet. Web services requests and responses are transmitted via Simple Object Access Protocol (SOAP) messages and their descriptions are defined in Web Services Description Language (WSDL). Users or programs, also called service requestors, use WSDL to determine what functions a Web service offers and what is required to access the service. Three operations are important in Web services – PUBLISH, FIND, and BIND. The service provider must PUBLISH the service so that it can be found by requestors. This service description is stored in a service registry, using UDDI (Universal Data Description Interface) directory technology, which allows the registry to be searched. The BIND operation allows the service to be used by the requestor, which could be a person or program. Once the service is located, SOAP is the protocol technology used for communication and is equivalent to an RPC call.

Web services architecture is illustrated in Figure 1.

## STANDARDS

Standards play a predominant role in identity federation by creating a common framework for Web services. Standards are designed to reduce the application development time, maintenance, and support overhead and achieve interoperability across organizations in the circle of trust. Proprietary methods of interoperability hinder integration and cross functional deployment, thus retarding Web services deployments.

There are several industry-standard development organizations, and consortia associated with ensuring the viability and enablement of Web services for federated identity. By creating specifications that can be widely adopted, heterogeneous systems and applications can communicate, thus creating an open environment. These standards are evolving but provide the basis for changing the network, allowing it to become an application platform. The current

generation of Web standards – Hyper Text Transfer Protocol/ Hyper Text Mark-up Language (HTTP/HTML) – must evolve to satisfy the new 'virtual enterprise'.

Several functional requirements are necessary for the successful implementation of Web services.

Standards organizations are:

- Internet2/Middleware Architecture Committee for Education (MACE)
- Organization for the Advancement of Structured Information Standards (OASIS)
- The Liberty Alliance
- IBM and Microsoft
- The World Wide Web Consortium (W3C)
- The Internet Engineering Task force (IETF)

### Internet2/MACE

The goal of the Internet2 Middleware Initiative (I2-MI), the project facilitated by Internet2 and a group of university middleware architects, is to assist in the creation of an interoperable middleware infrastructure among the membership of Internet2 and related communities.

Shibboleth is the initiative undertaken by this group. It is an open, standards-based solution designed to meet the needs of organizations that want to exchange information about their users in a secure and privacy-preserving manner. The organizations that have embraced Shibboleth to exchange information include higher education, their partners, digital content providers, and government agencies. Shibboleth uses OpenSAML, an open source toolkit based on Java and C++, for the message and assertion formats, and protocol bindings that are based on Security Assertion Mark-up Language (SAML).

## OASIS (formerly SGML Open)

The OASIS organization is made up of several technical security committees. It is a non-profit international consortium that drives the development and convergence of specifications based on public standards like Extensible Mark-up Language (XML) and Standard Generalized Mark-up Language (SGML). The focus of this organization is developing and ratifying e-business standards. Key technical committees whose work supports federated identity are:

- The Provisioning Services Technical Committee (PSTC). This committee produced the Web application security specification for Service Provisioning Mark-up Language, SPML, an open XML-based framework definition for the exchange of user, resource, and service definitions. This standard is designed to provide increased levels of interoperability between provisioning systems.
- The Security Services Technical Committee. This ratified the Security Assertion Mark-up language (SAML), another Web application security standard for the exchange of authorization and authentication information across domains. Systems across domains can now share a common XML-based framework for the exchanging of security assertions. Unless SAML is used, distributed resource management will be difficult to accomplish.

SAML = SOAP+HTTP – SOAP can be defined as simply a middleware protocol or an RPC that uses HTTP and XML for Web services. SAML is important for interoperability across heterogeneous domains and is made up of assertions (trusted statements), request/response protocols, bindings, and profiles. SAML has established itself as the foundation for Web services authentication. SAML specifications define SOAP headers for message authentication, message confidentiality, and integrity. Security tokens are used with digital signatures to ensure that only the recipient can unlock the message. Tokens supported are Kerberos tickets, and username and password. SAML is used for distributed

authorization in single sign-on, federated identity management, multi-vendor portals, and Web services access control. That is, it conveys identity to Web services.

SAML has been widely adopted and is rapidly becoming the *de facto* standard for federated identity deployments. Liberty has endorsed SAML for SSO implementations and to convey identity to Web services eg attributes, authentication, authorization; I2-MACE's leverages SAML in its Shibboleth specifications for SSO and attribute exchange. This SSO attribute exchange uses the SAML, SOAP/HTTP binding, and the attribute authorities are SAML compliant.

SAML is currently in an evolutionary state. The current versions in use are 1.1 and 1.2 but the major new version, 2.0, will integrate more closely with the Liberty Alliance federated identity standards. Liberty will input the features of ID-FF V1.2 in an effort to create a single standard for single sign-on, thus allowing the enlargement of the circle of trust, ie more than one-to-one relations can be accomplished.

The Web Services Security, or WS-Security, technical committee recently ratified Web services security specifications. The goal of the WS-Security specification is to improve interoperability between different security systems that use XML-based protocols, and define how authentication data provided by other security systems is transferred. This is expected to be used in a wide variety of products, including XML firewall products, Web services management software, and network access security products. IBM and Microsoft first authored this specification and then passed it over to OASIS and W3C for further development. WS-Security supports various security tokens, eg X.509 certificates, SAML assertions, username/password, and Kerberos tickets. This ability to support various security tokens is a major difference between WS-Security and the current Liberty Alliance specifications.

The XML Access Control Mark-up Language (XACML) technical committee is working on defining the core schema and namespace for the expression of authorization policies in XML against objects that are defined in XML.

The Extensible Rights Mark-up language (XrML) technical committee has ratified a generalized framework for enterprise rights management, or the ability persistently to protect enterprise intellectual property and promote its appropriate use. The intellectual property rights are owned by Content Guard, the vendor that submitted the specifications. XrML is used extensively in the music industry for music downloads over the Web.

Application Vulnerability Description Language (AVDL) was approved in summer 2004 to provide a standard way for exchanging information about security vulnerabilities between applications and Web services from different security tools. For example, the transport of data from a vulnerability scanner to a firewall to update the firewall policy protects the environment behind the firewall until a patch or other remedial action can occur. This is important to protect the integrity of data in a Web services environment.

The XML Common Biometric Format (XCBF) technical committee has approved the XML Common Biometric Format specification. This provides common automated methods for recognizing a person based on physiological or behavioural characteristics. The goal is to promote secure, interoperable, biometric applications and systems.

### Liberty Alliance

Liberty Alliance is a global consortium of over 150 vendors, established to create open, federated, network identity specifications, so that businesses can more easily interact while still maintaining security and privacy of information.

The Liberty Alliance has endorsed SAML specifications for the linking of accounts between domains to support SSO and Web Access Management, without divulging identity or attribute information. To that end, it has developed three specification modules, Identity Web Services framework (ID-WSF), the Identity Federation Framework (ID-FF) and identity services interfaces, all using SAML, HTTP, SOAP, and WSDL as the framework for supporting protocols and services.

ID-FF is a Web application security specification that extends SAML to define processes and schema for SSO, SLO, and name registration. It allows multiple circles of trust to federate in a secure environment with privacy protection.

The ID-WSF specification supports service discovery and the communication of service policy. It leverages ID-FF for principal authentication, federation, confidentiality, and privacy protection for messages and the service discovery. Security tokens are used for message-level authentication and authorization. The security tokens supported are SAML assertions and X.509 certificates.

Radio@aol, a service for accessing music and downloading to PCs, uses The Liberty Alliance specifications. Many other large organizations are using the Liberty specifications, eg General Motors.

## IBM AND MICROSOFT

The specifications by IBM and Microsoft, WS-Policy, WS-Trust, WS-Federation, WS-Secure Conversation, WS-Authority, WS-Privacy, and WS-Security, build on SOAP as a foundation or cornerstone, leveraging the already-installed systems of businesses, but enhancing their security at the message level.

WS-Federation is an attempt to build an overriding federated identity standard, to work in concert with SAML and other security standards. RSA Security, Verisign, and BEA have joined IBM and MS in this effort. This specification defines mechanisms that are used to enable identity, attribute, authentication, and authorization federation across different trust realms that use not onlySAML for authentication, but also Kerberos, Verisign certificate authority, or RSA authentication. The next generation of Windows XP, the much publicized Longhorn, will include Web services technology on a chip as part of its infrastructure technology, reducing the complexity of development and maintenance of applications; for example,

cryptographic functions, stored keys, and hashing to verify authentication

WS-Security technology is the most notable and it's finding its way into Web services software delivered by IBM and MS, BEA and RSA, and Verisign. MS Visual studio ships a toolkit as part of its Web services enhancements. It allows businesses to send messages with a digital signature to ensure that a document has not been altered during its transmission.

WS-Trust is a proposed standard for establishing secure communications that ensure document security, regardless of the security, eg Kerberos or PKI, enforced by the participating partners. Two other standards, WS-Conversation and WS-Policy, will make it easier to maintain security during multi-step transactions such as submitting an electronic purchase order over the Web.

Other proposed specifications, WS-Policy Attachments and WS-Policy Assertions, are designed to provide mechanisms that let businesses describe their security requirements as they relate to Web services applications, eg working with third-party authenticating services. This ensures integrity during the workflow and automates the management of policy communication and agreement. These standards have not yet been widely adopted.

WSDL is an XML-based language derived from Microsoft's SOAP and IBM's Network Accessible Service Specification Language (NASSL). WSDL is used to create service descriptions – about the location of the service, how to run the service, what business is used for hosting the services, and the kind of service it is.

## WORLD WIDE WEB CONSORTIUM (W3C)

The World Wide Web Consortium (W3C) also develops specifications and software to ensure interoperability on the Web. This international industry consortium is jointly run by the MIT Laboratory for Computer Science in the USA, the

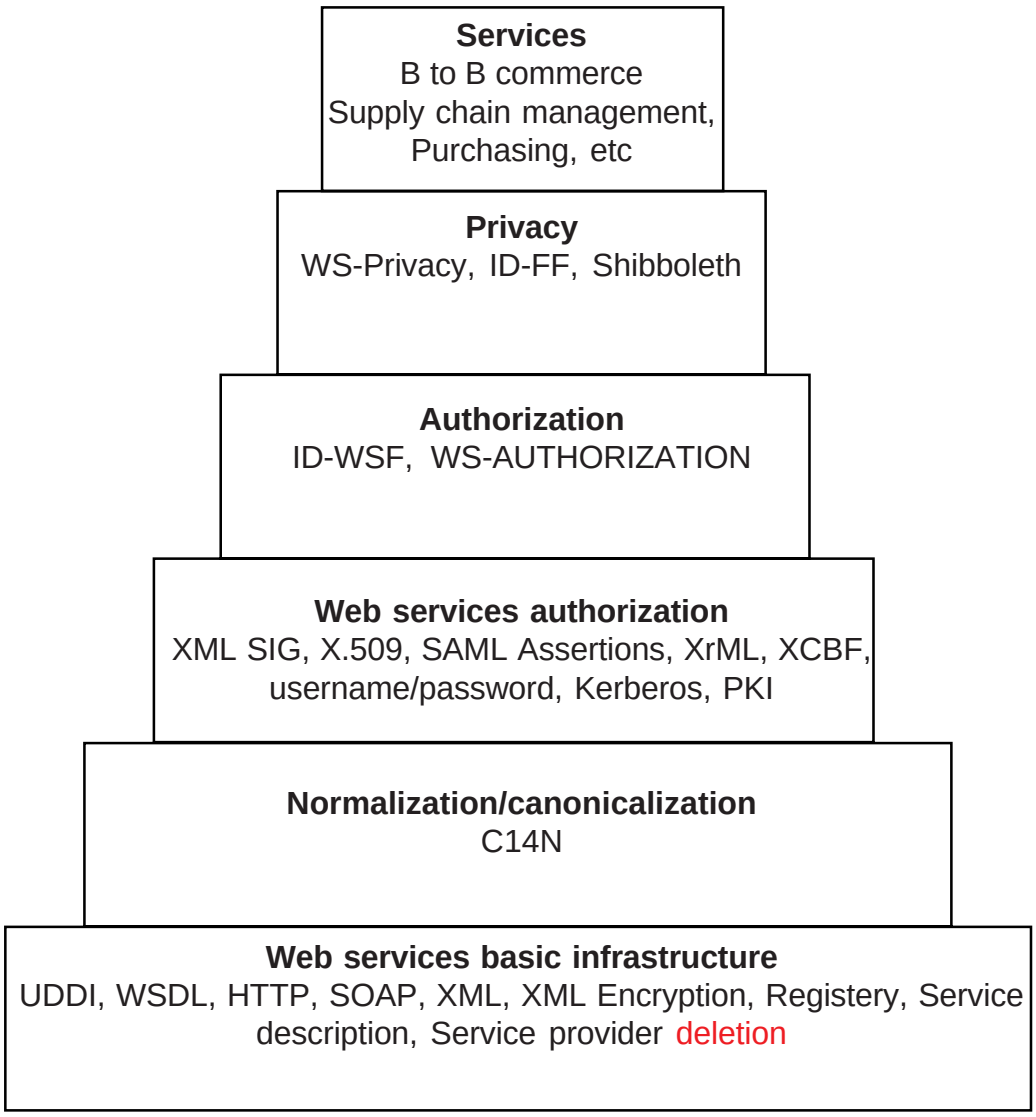


*Figure 2: Web services functional areas and standards*

National Institute for research in Computer Science in France, and Keio University in Japan. It is this group that delivered XML Encoding, the foundation for Web services computing.

To date the following successful activities have been accomplished and published:

- The XML Key Management System (XKMS) defines a set

of protocols for distributing and registering public keys. These protocols are suitable for use in conjunction with the proposed standard for XML Signature (XML-SIG) developed by the W3C and the Internet Engineering Task Force (IETF), and a developing companion standard for XML encryption. The XML Key Management Specification (XKMS) comprises two parts – the XML Key Information Service Specification (X-KISS), which defines a protocol for a Trust service, and the XML Key Registration Service Specification (X-KRSS). The X-KRSS specification defines a protocol for a Web service that accepts registration of public key information. Both protocols are defined in terms of structures expressed in the XML schema language, protocols employing SOAP V1.1 and relationships among messages defined by the V1.0 WSDL.

- XML Signatures (XML-SIG) provide integrity and message authentication, and/or signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere.

### C14N

XML documents that are logically equivalent within an application context may vary in physical representation because of the different syntax changes permitted by XML 1.0. The C14N specification describes a method for generating a physical representation, the canonical form, of an XML document that accounts for these permissible changes. Except for limitations regarding a few unusual cases, if two documents have the same canonical form, then the two documents are logically equivalent within the given application context. This makes document interchange and signature integrity possible.

### IETF

The IETF is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution and smooth operation of the Internet. The

community is made up of several working groups. The Web services technical work of the IETF is done in its security working groups.

IETF is responsible for Lightweight Directory Access Protocol (LDAP), the primary means by which directories are accessed over the Internet. Since the Web is layered over the Internet, it can be said that the protocols and specifications defined by the W3C are layered on those defined by the IETF. The IETF's PKIX working group is defining extensions and enhancements that cover Internet-based PKI. Figure 2 shows Web services functional areas and associated standards.

## SUMMARY

Web services provide a standardized way for dealing with the problems of distributed services and program-to-program communication. Federated identities can exploit Web services technology to create unique relationships outside the traditional organization. Companies can leverage data, partnerships, and, most of all, their customers in new ways, taking the use of the Internet and identity management to the next level. Federated identity technology allows individuals to use the same user name, password, or other authentication information for identification, to sign on to the networks of one, or more than one, enterprise in order to conduct transactions, regardless of the directory services, security, or authentication mechanism employed in the network.

## CONCLUSION

Federated identity exploits Web services; e-business is depending on Web services for it to fully exploit the value of the Web. Standards are the enablers of this new value model, since they create a common infrastructure for exchange of data. The adoption rate, however, has not been impressive. Some testimonials will lead one to believe that the assimilation rate in user environments is soaring. However, according to Dan Blum, Burton Group senior VP, the specifications are

young, and coupled with internal business pressures there have been only 200 production implementations of SAML.

There could be several explanations for this, the major ones being sorting out the standards between competing organizations, auditing, and trust. There are conflicts between OASIS and W3C, Liberty Alliance and MS, IBM, *et al*, with the two major competing standards being WS-Federation from IBM, MS, BEA, RSA, and Verisign, and Liberty (ID-WSF and ID-FF) from the Liberty Alliance. Even though they both have embraced SAML, there are some fundamental differences in terms of functional area decisions, roadmaps for delivery, and authentication mechanisms. Regulatory compliance has forced organizations to audit their environments. Currently there are no central robust mechanisms for auditing an identity as it traverses domains. This issue may be viewed as a high-risk factor in some environments. In addition, there is the issue of trust between organizations. Unfortunately, trust cannot be legislated or built into IT, and the many legal and business issues will be novel concepts to organizations.

Which standards will survive? I believe that time will tell. The customers will determine where this technology goes and which standards will endure. The ideal environment will have many solutions to choose from, since anything else stifles creativity. However, as long as the underlying framework remains the same, there is no 'Darwinian' phenomenon in the future for Web services and its support of federated identity.

Part 2 will focus on business drivers, implementation challenges, and best practices in deployment.

*Annette Hagood*
*Director, Product Marketing*
*eTrust Identity and Access Management (USA)*

## CICS and WebSphere

Getting CICS and WebSphere to interoperate was always a tricky task. IBM's recent announcement should help with this process.

The CICS Business Event Publisher, currently at Version 1.2, is basically a tool to allow CICS to drive business processing in a mixed workload environment. And that workload can include other CICS systems, WebSphere MQ, WebSphere, WebSphere J2EE, IMS, and WBI Message Broker. Business Event Publisher allows users to track events that occur in CICS, and through a rules-based engine drive transactions with WebSphere or with CICS.

What some of IBM's claimed 10,000 CICS customers are trying to do is allow WebSphere applications to drive their established and secure CICS transactions. In addition to that, CICS transactions sometimes need to collect information and/or update information on other systems. Business Event Publisher allows CICS to 'know' where to drive outbound business processes. This is achieved by the Business Event Publisher making it possible for CICS to query WebSphere or a WebSphere database (or whatever) to get the required information, operate on it, and send out a response or an update. It can even drive a DB2 stored procedure. In effect, Business Event Publisher allows CICS to have a more direct outbound business process-driving capability. And this is the bit that was so hard to do before the product became available.

Also recently announced was the CICS VSAM Transparency product, which supports a mixed workload environment between WebSphere and CICS. With VSAM Transparency, users can migrate data from VSAM to a DB2 database. It's useful because it means that WebSphere J2EE application developers can now more easily access the data and there's no need to re-code all the existing CICS applications that use the data. In essence, CICS thinks the data is still in VSAM

when it isn't, and WebSphere applications use the same data in a new DB2 database. The advantages are that there is only one copy of the data, there is no need to re-code the existing CICS applications, and the total cost of the new WebSphere applications is therefore much less.

Also new is CICS VSAM Copy (new for IBM that is; Computer Associates and BMC already have this kind of product). This product allows CICS files to be copied at the same time as they are open for any updates. In the past, copying a VSAM file meant making it unavailable to other applications, which in a CICS environment really meant taking CICS off-line while copies were made. In an environment where WebSphere is being used to drive CICS transactions, taking CICS off-line means that WebSphere is badly affected. Using VSAM Copy means that CICS will no longer have to be taken off-line, and consequently the WebSphere application will no longer have to go off-line.

Last in this group of announcements is Version 1.3 of CICS Performance Analyzer, which now has hooks into the WebSphere and MQ management infrastructures (plus other features). The new release has: historical data capture and storage; statistical analysis; and integration with the other performance analysis products within IBM (particularly the TDS 390 product from Tivoli).

IBM has now made the integration of its new WebSphere product line, with its venerable and still hugely successful CICS product, much easier for users.

*Nick Nourse*
*Independent Consultant (UK)*

IBM has announced Version 3.1 of CICS TS for z/OS, which adds Web services technology to extend CICS applications to an SOA environment. IBM also announced CICS Transaction Gateway V6, which supports Java 2 Platform, Enterprise Edition (J2EE).

CICS TS 3.1 introduces a flow management capability that will enable organizations to orchestrate CICS applications in short-running microflows that can be published on an enterprise service bus via Web services, WebSphere MQ, IIOP, and other protocols.

For further information contact your local IBM representative.
URL: www-306.ibm.com/software/htp/cics/tserver/v31.

* * *

Nastel Technologies has announced Version 4.0 of AutoPilot/Transaction Monitor, its transaction monitoring module.

AutoPilot/Transaction Monitor 4.0 automatically tracks transactions and messages that flow in the WebSphere MQ network end-to-end, and uses a message interception facility to ensure smooth message flow and minimal impact on the performance of an application. The product enables users to monitor, record, track, and troubleshoot WebSphere MQ messages as they are transmitted across a network in real time.

A user is able to: monitor messages and check the status at any time of any message sent; find a particular message at any point in the network; detect compromised or incomplete transactions; provide precision and granular information on any anomaly and performance bottlenecks; plan for future growth, detect patterns, and measure capacity.

For further information contact:
Nastel Technologies, 48 South Service Road, Melville, NY 11747, USA.
Tel: (631) 761 9100.
URL: www.nastel.com/news/new_pr37.shtml.

* * *

Forum Systems has announced Version 4.1 of Forum Sentry, its Web services security gateway. Forum Sentry 4.1 has evolved from securing Web services within service-oriented architectures to protecting event-driven applications.

New technology advancements include secure message and protocol intermediation, which streamlines the adoption of event-driven applications using XML and Web services while leveraging existing investments in enterprise message-oriented middleware.

Forum Sentry 4.1 is able to secure many-to-many asynchronous interactions using existing Message-Oriented-Middleware (MOM) such as IBM WebSphere MQ, Tibco Rendezvous, and JMS, and message-level XML and Web services authentication, authorization, and routing.

For further information contact:
Forum Systems, 45 West 10000 South, Suite 415, Sandy, UT 84070, USA.
Tel: (801) 313 4400.
URL: www.forumsys.com/products_sentry.htm.

* * *