70

MQ update

April 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.

z/OS and WMQ performance information

WebSphere MQ for z/OS provides facilities that collect statistics and accounting information and record the information in SMF records.

To identify performance problems, it can be useful to analyse the data found in SMF type 115 records. Other SMF records that can help identify problems are the accounting information in type 116 records and Coupling Facility Activity information in type 74 records (which can help to identify performance problems with shared WebSphere MQ queues).

The areas where problems can occur are:

- Queue manager – there may be problems with the basic queue manager parameter specifications, assignment of queues to page sets, assignment of page sets to buffer pools, applications commit activity, and incorrect queue index specifications.
- Buffer pool manager – there may be problems with the use and availability of buffers in each buffer pool, the size specified for specific buffer pools, the I/O activity to pagesets on DASD, synchronous and asynchronous write activity (both exceeding thresholds and in-use pages during commit), and the characteristics of messages assigned to buffer pools.
- Log manager – there may be problems with backout activity, allocation of active logs, messages written to active logs, messages written to archive logs, and checkpoint activity.
- Server – there may be problems with DB2 server calls, DB2 server delays, DB2 server deadlocks and ABENDs, and other areas that indicate DB2 performance problems.
- Shared queue and shared queue cluster – problems are identified by examining SMF type 74 records, although

some initial evidence will be provided by type 115 records. Coupling facility structure definitions and usage for WebSphere MQ shared queue structures will be where the problems lie.

Stephen Hare (USA)

Going beyond event-based MQ monitoring

Every business transaction is a process comprising related, interdependent, and often highly-complex events. A breakdown within any event, at any point in the transactional process flow, may set off a chain reaction that can disrupt business operations and cost companies millions.

Ensuring the availability and reliability of transactional processes is crucial for today's real-time business. At a minimum, this requires on-demand visibility into business application process flows from end-to-end across the entire IT infrastructure. Real-time event-based monitoring tools can provide a degree of insight into transactions and the complex events that comprise them start-to-finish.

To fully ensure near-zero latency, however, just seeing what's happening right now isn't enough. What you need is the equivalent of a crystal ball that lets you see what's coming in order to take pre-emptive action before failure strikes. While there may not be actual crystal balls out there, today you can get close to this kind of predictive ability by going beyond simple event-based monitoring.

Going beyond event-based monitoring means re-thinking the way events are defined and generated, and leveraging new technologies that allow events from WebSphere MQ to be dynamically correlated in a variety of ways with events from other systems across the application infrastructure – such as

other middleware platforms, Enterprise Systems Management tools (ESMs), databases, Web and application servers, and even operating systems.

A SOLID FOUNDATION

Event-based monitoring has long been the foundation for monitoring the individual technology components of the complex infrastructures that support the flow of integrated enterprise application processes. This approach relies on a predefined set of events, which are generated from each IT device or technology component in the IT infrastructure, such as WMQ. These events can in turn trigger the generation of other preconfigured events or alerts based on certain conditions being met, thresholds reached, or rules applied.

In this model, each event is focused solely on the IT device or technology that generated it, giving a view of only one narrow segment of the overall infrastructure. Component-specific monitoring tools track these events to provide insight into the status of processes within that discrete component of the infrastructure.

However, the complex heterogeneous IT infrastructures that support today's integrated enterprise business applications typically incorporate a wide variety of disparate technology components, ie WMQ, Oracle databases, an ERP system like SAP, an ESM (like IBM's Tivoli, HP OpenView, or CA Unicenter), a network communications platform such as TIBCO, operating systems (including Unix, Windows, Linux, and various mainframe operating systems), assorted Web servers, etc. Each one may generate thousands of static events and alerts, which could, in a complex environment, spawn even more.

It's easy to become inundated with too much information. The problem is exacerbated because many of the events and alerts are extraneous or non-mission critical given other concurrent IT environment-related conditions. Consider the volume of events and alerts generated by WMQ alone.

Thousands of staff-hours are spent dealing with the clutter of non-essential alerts, but the effort required to get to the root cause of the problems increases because of the lack of useful information provided. When you multiply that by the number of other infrastructure technology components in a typical enterprise application environment, the number of hours consumed by weeding through events and alerts is staggering.

Too much information is only part of the problem, though. While all the pre-configured events that are churned out may offer accurate data about individual components of the infrastructure, they can't provide meaningful insight into the health and reliability of the overall infrastructure, nor the start-to-finish integrity of the transactional processes that traverse it, because they show only the status of processes within their single technology component scope. So, while event-based monitoring established a solid foundation, it is one with inherent limitations when it comes to gaining the consolidated 'big picture' end-to-end view that is required for taking proactive infrastructure management to the next level.

PREDICTIVE FAULT PREVENTION

In the simple event-monitoring paradigm, the event itself is the primary element. In reality, though, the primary element is, in fact, the metric or measurement, which may be technology related but may also be business related. After all, the primary function of IT is supporting business processes and the users whose job functions rely on those processes. It's not enough simply to alert a front-line employee about a breakdown. In the real-time business environment, the alert needs to be framed in the context of the user's business function so that s/he can respond appropriately – and immediately.

The main goal of event-based monitoring is rapid problem detection. While this is fine as far as it goes, it provides business users with only pre-defined failure notifications, usually after the service is already degrading, disabled, or unavailable. Further, it can entirely miss problems that are

caused by a convergence of disruptive conditions, bottlenecks, disconnects, or failures in the interactions between infrastructure components because these aren't visible with component-centric monitoring tools. This can result in non-compliance with Service Level Agreements (SLAs) as well as a loss of worker productivity, increased cost to the business, and potentially lost revenues.

To predict and prevent performance problems, rather than simply detect the failure after it occurs, it's vital to understand how a business process behaves and to track performance trends associated with that behaviour. By examining the performance metric trends over time, it's possible to determine the thresholds and conditions that presage the likelihood of process degradation and failure. The ability to collect, correlate, view, and analyse these performance metrics from the many disparate systems across the infrastructure is the key to gaining the deeper level of understanding that is required.

There are many event-based monitoring tools available today for each of the most commonly deployed infrastructure components, like WMQ. These component-specific event-based monitoring systems do a very good job of capturing events generated by the system they're designed and built to monitor. But typically they aren't equipped to capture and correlate events generated by external sources (eg other software components or IT devices like databases, ESMs, application servers, etc) with events generated by the monitored component. Achieving any degree of cross-platform process visibility is extremely complex and typically requires hardwired event handling and sophisticated programming event-by-event.

Another limitation of most event-based monitoring systems is that they are too far removed from the primary or business metric. After all, they're designed to monitor technology functions that may or may not map one-to-one (or at all) with the business processes they support. Business processes are made up of a series of transactions, which in turn are made

up of a collection of complex events that may occur over a wide variety of technologies and infrastructure components.

IT LOOKS SIMPLE ON THE SURFACE

In an example of a relatively simple real-time business transaction, a business user may request a line of credit inquiry. This can involve:

- Validating the customer's existence in a DB2 database on the mainframe, which in turn triggers:
  - messages to be sent via WMQ to a SQL Server database that stores archival customer data to retrieve the customer's historical credit rating.
  - messages to the accounts receivable application running on a Unix server to retrieve current account balances and recent payments recorded in an Oracle database.
- Delivering all these values to the credit processing application, running on a Windows platform, which calculates the available line of credit in real time.
- Returning the result to the business user's desktop.

This seemingly simple business process transaction requires over 30 discrete interactions to be successfully completed, including numerous inter-system messages being generated, launched, and delivered via the WMQ middleware, and multiple database accesses and calculations successfully executed by various applications on five different hardware platforms running three different operating systems.

All of this is transparent to the business user – he wants just the results he needs to give to the customer. But if the transaction is painfully slow in delivering the results to the business user's desktop, is it because the server running the credit application is down, or because the Oracle database is inundated with table access requests because the accounts

payable department always processes its main cheque run on this day of the month, or is it because the initial WMQ message to the DB2 database on the mainframe for the customer ID validation is languishing in a message queue somewhere?

In the event-based monitoring environment, there would need to be an event-related alert for each of the 30-plus discrete inter-system interactions in order to instantly know the answer to that question. That's why it's so difficult and time-consuming to tailor alerts to meet business users' needs. It's also too time-consuming and labour intensive to accommodate the real-time pace of business today.

To gain the business metric perspective that will allow predictive fault prevention requires:

- A consolidated view of business processes end-to-end.
- Drill-down visibility into the actual or root cause of problems.
- The ability to see and interpret the effect of events on the business.
- The means to capture historical trends and map them to current activities to predict behaviour changes.
- The agility to adapt quickly.

UNDERSTANDING THE BUSINESS IMPACT

Understanding the business impact of systems requires viewing them from a holistic, enterprise-wide, business perspective – how the systems fit together and interoperate to support the business. The monitoring technology that will provide the necessary understanding of business processes must view enterprise systems in terms of how they're used, rather than in terms of how they're built.

Event-based monitoring, by its nature, can view systems only in terms of how they're built – because it can view only the individual component building blocks, not the entire structure.

It requires manually combing through all the different components' log files to find the intersections of alerts that represent the root cause of a problem. Plus, there's typically so much data that it can be virtually impossible to see the relationship between events, or determine which combination of events may share a common cause.

To see systems from the business perspective requires monitoring and management technology that has the ability to automatically harvest and correlate deeply granular data from a wide variety of sources and apply any possible combinations, permutations, and rules.

DYNAMIC EVENT CREATION

Business perspective monitoring also requires the ability to dynamically create new events based on business-relevant criteria. These events need to be user-definable to map to individual business users' informational needs. They will typically comprise related events and alerts generated from various systems, and should have the ability to incorporate user-defined business rules, thresholds, conditions, and filters. Business perspective monitoring technology must also have the ability to turn correlated results and dynamically-generated events into actionable policies that deliver precise and relevant information while simultaneously invoking corrective actions.

In the line of credit transaction example described above, monitoring technology operating from the business perspective would be able to gather process status and performance metrics simultaneously from WMQ, the mainframe OS, the DB2 database, the Unix server, the SQL Server database, the Oracle database, and the Windows systems, and to correlate them in real-time to provide visibility across all the systems involved.

When the relevant events and alerts from WMQ are correlated with related events from the other systems involved in the process, the root cause is instantly visible. If the problem were, in fact, the WMQ message stalled in the queue, the

monitoring and management technology should also have the ability to automatically launch the appropriate corrective measures, whether they be re-routing the message or re-starting the queue manager, to enable the completion of the transaction.

To enable predictive prevention of the same problem in the future, the symptoms and conditions that triggered the problem can also be correlated and used to anticipate and pre-emptively preclude an impending problem of the same nature.

BUSINESS AGILITY

Business needs change quickly and constantly. The monitoring of business-critical metrics must continually adapt to changes as they occur in order to remain effective.

The right business perspective monitoring technology, with event correlation and dynamic event creation capabilities, enables adaptations to be made quickly and easily, even to processes in live production, to keep pace with changing business processes. Together, these capabilities provide the equivalent of a crystal ball to let you see not just what's happening now, but to predict and pre-empt costly system failures and ensure business agility.

David Mavashev
CEO
Nastel Technologies Inc (USA)

Why not share your expertise and earn money at the same time? Send us an article. When we publish it, we will send you a cheque, as payment, and two copies of the issue containing the article. E-mail the editor, Trevor Eddolls, at trevore@xephon.com.

WebSphere MQ performance

When it comes to deciding which areas are the best ones to look at as potential ways to improve performance, it usually boils down to a common set, even though there could be a number of different platforms involved, from a z/OS mainframe, almost any sized Unix box, to smaller Windows machines.

NETWORK ISSUES

It doesn't matter how fast your processor is or how much disk space is available on the different platforms, if your network is slow then WMQ will run slowly. The three key areas you're interested in here are:

- How fast is the network?
- How much traffic does it have?
- How large are the messages you're trying to send?

The simple way to increase network speed is to buy higher-speed networks. Internally this is a possible solution, particularly as the prices seem to decrease.

You could always try to reduce other network traffic! Alternatively, you could try to reduce the size of the messages being sent. A number of vendors supply software that can help with this.

An application that expects messages from a remote queue manager that uses a temporary dynamic queue for the reply queue may cause performance problems if it terminates before the response is received. That can create a dead-letter situation. With dead-letter processing, the normal channel flow is disrupted. WMQ clustering can eliminate many common causes of dead-letter failures, including queue not found and queue disabled.

The batchint channel tuning parameter can affect performance.

MQ batches up messages and then sends them. Batching them up reduces the amount of channel processing required (so that's a good thing). Typically, if the associated transmission queue is empty, MQ will end the batch. However, you might want it to wait a little longer so it can send a larger batch – it depends on the message arrival rate. So, using the batchint parameter, you specify that MQ should wait until the specified batch size is reached before sending the batch. However, if the message arrival rate is slow, using this parameter can cause delays.

PROCESSOR ISSUES

The type and number of MQI calls issued impact on the processor consumption of WMQ applications.

The MQI calls are shown below with the largest consumers of CPU resources first:

- MQCONN – connects to the queue manager and creates the required task structures and control blocks.
- MQOPEN – opens a specific queue for processing, may lock the required resources, and acquires control blocks.
- MQCLOSE – closes the queue, commits resources, frees locks, and releases control blocks.
- MQPUT – puts a message on a queue (recovery processing may be required).
- MQGET – gets a message from a queue (recovery processing may be required).

The simplest way to reduce usage of the CPU is not to make MQI calls, wherever possible. There are a variety of ways of doing this, but the most appropriate solution usually depends on the arrival rate of messages.

It's also a good idea, where possible, to reduce the size of messages or to compress messages before they are placed on a queue.

Another approach is to cut down the number of messages sent. If an application is producing a number of messages fairly frequently, it may improve performance to combine them into one, single, larger message.

Performance improvements can also be realized by using intermediate commits for large numbers of messages. Using periodic commits will make the messages available to the server application, rather than their all appearing at once with a single commit. It also reduces the impact on other applications.

The general consensus of opinion is that sharing queues within an application domain can be good for performance.

I/O ISSUES

It doesn't matter how fast the processor is – if an application is waiting for a disk I/O to complete, the performance will be poor.

One of the main I/O concerns with WMQ is logging. So that messages are delivered, and delivered only once, every message must be logged. And WMQ must ensure that the log has been committed prior to the work unit's completion.

Logging is performed only for persistent messages, so the use of non-persistent messages eliminates logging activity. The downside, of course, is that non-persistent messages may never be delivered. Non-persistent messages are not maintained across a restart of the queue manager. However, there are good examples where non-persistent messages can be used.

Smaller or compressed messages need less buffer space to hold the message, both for message data and for logging.

For large numbers of messages it's best to perform intermediate commits. If the number of messages within a unit of work increases, WebSphere MQ will be unable to keep all the data

in internal buffers and will have to write some of it to disk, which has a negative impact on performance.

Separating logging volumes from data volumes improves performance by reducing contention on the disks. It also means that there is not just one single potential point of failure.

CONTENTION ISSUES

A variety of factors can cause contention issues for WMQ applications.

WMQ supports multiple servers per queue. If you find that messages are arriving faster than the server can process them, the solution would be to add servers.

If the number of servers required is greater than the platform's capacity then the solution is clustering to provide servers across multiple systems. This requires that the application be able to process across multiple servers.

CONCLUSION

There are a number of areas that contribute to the overall performance of WebSphere MQ. A closer examination will help you identify where to concentrate your resources.

Nick Nourse
Independent Consultant (UK)

Message validation with WebSphere Business Integration Message Broker

This article gives an overview of message validation. It provides a general discussion of different kinds of message validation and how to implement them with WebSphere Business Integration Message Broker (hereafter called WebSphere BI MB).

INTRODUCTION

WebSphere BI MB is a message broker that deals with application-defined messages. For many scenarios it is necessary to validate the messages and their contents before processing them, after creating them, or before forwarding them to another application. Message validation in WebSphere BI MB has been examined as part of the WebSphere Business Integration for Financial Networks (abbreviated to WebSphere BI for FN) product development (for details see http://www.ibm.com/software/integration/wbifn). WebSphere BI for FN consists of a base part providing functionality and services that can be used to more easily create and deliver products on top of WebSphere BI MB. In addition there are multiple product extensions that provide value-added functionality. For example, the Extension for SWIFTNet provides access to the Secure IP Network, a private network operated by SWIFT (for details about SWIFT see http://www.swift.com).

SWIFT not only provides the private network, it is also a standards body that defines standards for messages interchanged between financial institutions. One such standard is the FIN standard covering many different messages for various financial operations – for example payments and securities.

MESSAGES AND MESSAGE STRUCTURES

Physically, a message is a series of bytes. It is the information that is exchanged between applications. The messages are usually interpreted by application programs. To allow this, it is assumed that the message has a logical structure that is defined as an interface between the applications. There are different ways to describe the structure of a message, for example C or COBOL data structures, tagged data structures, or self-defining data structures like XML, which is described by Data Type Definitions (DTDs) or XML schemas.

The data structure usually defines the elements in the byte

stream and their logical hierarchy. There can be different kinds of element within the structures. Elements can be mandatory or optional. There can be single elements, lists with a defined number of elements, or lists with an undefined number of elements. An element may also be a structure of elements. With this definition, a message in general can logically be seen as a tree of message elements.

Within a data structure, any element usually also defines which values can be stored in it. Some ranges for an element may be defined by the type of element used, for example a 2-byte positive binary number ranges between 0 and 65,535. The message definition may further reduce the possible range.

An actual message then has values for each of the elements in the structure. The task of message validation is to verify whether this message content conforms to the defined format described by the structure. Such data structures are defined either by the application or by standards organizations.

DIFFERENT KINDS OF MESSAGE VALIDATION

A message can be validated against different criteria. These can be categorized into the following kinds of message validation:

- Structural validation
- Field content validation
- Cross-field validation
- Validation against directory information.

Which validation is actually required depends on the business problem that you need to solve. The following sections describe each of the message validation forms.

Structural validation

The task of structural message validation is to check whether

the message conforms to the physical structure as defined in the message format. There are different ways that such validation can be performed. The simplest form for structural validation is to check the length of the data. It is simple for fixed-length data structures, like those defined using C or COBOL structures. Depending on the elements in the data structure, each element may have to be checked to get the total length.

Some tag-delimited data structures may have defined field delimiters – begin tags, end tags, and fill characters. Structural message validation checks whether these elements in the byte stream are at the defined place and have the correct constants.

Further validation can be to check whether all the required message elements are available, or whether there are elements in the data that are not allowed. If the structure contains lists of elements and the number of elements is dynamic, it can be validated that the number conforms to the allowed range.

Field content validation

Field content validation checks the value of all the fields in a message. An individual field can have different restrictions defined:

- A predefined value – this is like a constant or a delimiter within the message. The values depend on the data type of the element, for example a number or a character constant. This kind of data is usually used to identify the type of the message, or its version number.
- A predefined list of values – there are many fields in messages that are allowed to carry only a value from a list of predefined values. Depending on the data type of an element, this can be character constants or number constants. For example, an integer field may be limited to the values 1, 3, and 7.
- A predefined range – this is similar to a predefined list, but

it is a simplification to express a set of allowed values, usually for numbers; for example, the content of an integer field must have a value in the range from 1 to 13.

- A predefined pattern – for character fields, this is a common method to describe an open list of elements that have to conform to a standard, for example a value that consists of only characters. More sophisticated patterns are usually needed for other common data structures like a date, time, or timestamp values.
- A check for correct data type – in many messages that are exchanged between applications, numbers or binary data are encoded in characters. A simple check for the correct data type is usually the test whether a character string represents an integer. Usually, this could be a combined test with a predefined pattern, for example five digits and a range check – a value between 0 and 65,535 for a positive 2-byte integer.

To allow these checks, the message must first conform to the structure. The field content validation is then performed on the logical level of the message.

Cross-field validation

Often the allowed value of one field depends on the content of other fields. There are different ways that such dependencies could be defined. Examples are that one element is allowed only if another field has a specific value. There are also situations where one field has to contain a value that is calculated based on other fields, for example one fields has to contain the sum of all the values within a list. A third example is a field that contains the number of elements within a dynamic list of elements.

Cross-field validation checks that such kinds of dependency between different fields, potentially in different hierarchy levels of a message, are satisfied for a message.

Validation against directory information

Some message standards contain elements that are allowed to contain values, which have to be in a list of constants published elsewhere. Examples are currencies or country codes. The allowed values for these elements are usually defined by an international standard, for example, the International Standards Organization (ISO) (for details about ISO and ISO-defined standards see http://www.iso.org). In other situations, the values have to be in customer-managed databases. For example, a message can contain the trading partner of the originator of the message. In this case the trading partner has to be in the list of allowed trading partners for the originator.

For validation against such data it is necessary to know the location of the information and how to access it. There are many possible locations for such directory information, for example a flat file, a database table, or an LDAP directory.

MISCELLANEOUS MESSAGE VALIDATION REQUIREMENTS

Beside the different kinds of message validation, there are other things that have to be taken into account when working on a message validation solution. These items are:

- Validation standards
- Error codes
- Complete validation.

These topics are further discussed in the following sections.

Validation standards

Some message formats, like the SWIFT FIN format, are defined as an international standard. Parts of this standard include message validation rules. These rules define, for example, which fields are optional. Such standard messages are also used in different user groups, or within a company for

internal communication purposes. In some cases like this, extended validation rules apply. Such additional rules are usually stricter than the standard rules, for example the user group can make a field that is optional in the standard mandatory for its own purposes.

This means that for validating messages it is not sufficient to have one set of validation rules for all messages. Instead, during the processing of any message there must be a criterion to determine which set of validation rules has to be used for that message. Such criteria can be coded, as in cases where one message flow processes only one kind of message, or the message must be enriched – for example, with information in some message headers that allows you to determine whether it is to be validated against the standard rules set or a private rules set.

Validation error codes

If the message validation processing concludes that a message does not conform to the validation rules for any reason, an error has to be returned to the originating user of the application. In most cases, the error codes that have to be issued can freely be defined. But in the case of validation rules in a standard, the standard sometimes also defines the error codes that relate to the rule that is violated. Usually an error code uniquely identifies the incorrect part of the message and informs all users what is wrong with the message (without them needing to know about the application). This allows the error code to be communicated to other applications, even if these applications belong to a different organization or company.

Complete validation

Most messages are sent by applications and you can be sure that only a few of them, if any, contain errors. But there are situations in which messages are inserted manually, for example if a message of a certain type is sent only in error

situations or very rarely (for example the message once a month).

In these situations, a message can contain not only one error but multiples of it. If this is the case, the message must be corrected. Message repair in most situations is done manually. To be able to correct the message at once, all errors within the messages should be reported. Otherwise the same message may come to the message repair function more than once. This kind of manual processing is usually very expensive and takes a long time.

WEBSPHERE BI MB MESSAGE VALIDATION

Message validation capabilities were provided from the beginning of the WebSphere MQ Integrator Broker product, the predecessor of WebSphere BI MB. It is steadily improving with each new version and sometimes also within one release. Nevertheless, it still does not offer full message validation capabilities as described in the previous sections. This section describes what message validation capabilities are already built into the WebSphere BI MB product.

Message sets

The structure of a message in WebSphere BI MB is defined by creating a message set. Such a message set can be defined from scratch. But it is also possible to import definitions, for example, from C/C++ header files, COBOL copybooks, XML DTDs, or XML schemas. A message set describes both the logical structure of the messages and what they look like as bit streams (their physical structure).

A message set can contain descriptions of one or more messages or formats.

Message set validations

Once you have defined your message set and you receive a message that should conform to one of the message formats

in the message set, you have different ways to validate the message. The first and simplest one is to access the message using the logical tree. To allow this, the parser in the broker reads the message set and uses this to convert the bit stream of the message into the logical message tree. If the parser is able to do this, the message conforms to the physical structure defined by the message set. WebSphere BI MB delivers a set of different parsers, called message domains. This set includes parsers for most WebSphere MQ formats, MRM, XML, and others.

Be aware that the parser usually parses only up to the element in the bit stream that corresponds to the element in the logical message tree that you are accessing. In some situations this can result in invalid messages passing the system. To be sure that the complete message conforms to the definition in the message set, you have to access the last element in the message. This is usually very easy. Either you know the structure of the message completely and know which element to access, or you can write a small algorithm to navigate to the last element in the message. A simple implementation in ESQL could look like:

```
DECLARE LAST_PTR REFERENCE TO InputRoot;

WHILE (LASTMOVE (LAST_PTR)) DO
      MOVE LAST_PTR LASTCHILD;
END WHILE;
```

The same algorithm also works for self-defining data structures, like XML. In such a case, no message set is strictly required. If the last element can be accessed, the message conforms to the XML standard and is called ‘well formed’.

Besides the information about the physical structure in the message set, you can also define most of the restrictions on single fields, as discussed for message content validation. WebSphere BI MB can also perform validations to ensure that such restrictions are met. To allow this, you need to turn on validations. For message flows started by an MQInput node, this is done by setting attributes in the validation tab of the

properties dialog for this node. Most other input nodes have similar properties.

By using properties in the validation tab, you can decide, for example, whether validation needs to be performed at all, what actions should be performed if a message doesn't comply with the message set, whether messages need to be checked before leaving the input node, or whether the message should be validated while you access elements in the message flow. For a detailed list of validation properties, their possible values, and their meanings, refer to the WebSphere BI MB documentation.

If the parser cannot match a message against the message set or the parser detects other kinds of problem, for example tags not matching correctly, the parser throwing a WebSphere BI MB exception. Such an exception consists of a WebSphere BI MB error number, for example BIP5431. The message text and the associated parameters usually provide sufficient information for a programmer to understand what's wrong with the message. There are no ways to define error codes for such predefined situations. A mapping of the parser exceptions to the required validation error codes has to be done within the message flow, for example, in a Compute node that is connected from the catch terminal of a Try-Catch node. Whether a complete mapping of WebSphere BI MB exceptions to any standard defined error codes is possible depends greatly on the complexity of the message set and the kind of problem that is in the message.

Errors that result from message validations against a message set behave in the same way as the matching problems. Once stopped, there is no way to let the parser continue with parsing and validating the message. In cases where the message contains more than one problem, the second problem cannot easily be detected before the first problem is resolved.

With properties on the input nodes, for example the MQInput node, you can define the message set that should be used to parse and optionally validate the input message of the message

flow. If you have to validate a message according to different standards and the validation capabilities of the provided WebSphere BI MB parser are sufficient to fulfil the requirements, there are different ways of overriding the message set that is used.

The first method is for the application sending the message to provide the information about the message set that has to be used for parsing and validation. For this, the application has to provide the corresponding information in the message content descriptor (mcd), a WebSphere BI MB-defined folder in the MQRFH2. The mcd information overrides any message set information provided in the MQInput node.

Usually an application does not know about the broker or it doesn't provide an MQRFH2. In a message flow, you can choose a different message set for parsing the message. One way to change the message set is by using a ResetContentDescriptor node. With this node you can force the broker to use another message set or format. But the new parameters have to be set statically as properties. If you have to support multiple message sets you need multiple ResetContentDescriptor nodes and some processing to determine which one to use. Another drawback of this node is that you cannot force validation of the message.

Dynamically parsing a message and forcing message validation can be done in Compute nodes using ESQL. For example, the statement:

```
CREATE LASTCHILD of OutputRoot Domain MRM
PARSE (InputRoot.Blob.Blob
SET YourSet
TYPE YourType
FORMAT YourFormat
OPTIONS ... );
```

allows you to parse a message that is received as binary large object (BLOB), saying no message set is associated with the message. Using the SET, TYPE, and FORMAT options of the PARSE clause you can define which message set and format to use for parsing. These parameters allow variables to be

used, so you could choose the message set that suits the messages. With the OPTIONS option, you can define whether, when, and how the message is validated in this process. For details on the PARSE clause, its options, and possible values, refer to the WebSphere BI MB documentation.

Both possibilities, the ResetContentDescriptor and the ESQL CREATE path, require that you have a criterion to decide which message set and format to use. Such information could either be retrieved from the content of the message or passed in from the application.

EXTENDED MESSAGE VALIDATION WITH WEBSPHERE BI MB

The message validation capabilities built into WebSphere BI MB end with message content validation. Cross-field validations and validations against directory information are not supported by the standard parsers. But this kind of message validation can easily be implemented with standard WebSphere BI MB nodes, for example the Compute node. In ESQL you can code most of the tests. For instance, the next ESQL code fragment shows how to validate a simple dependency between two fields:

```
If InputRoot.MRM.MyStruct.AField = "ABC" THEN
      IF InputRoot.MRM.MyStruct.Field2 IS NOT = "XYZ" THEN
            /* Illegal cross-field dependency detected */
            ...
      END IF;
END IF;
```

In cases where the validation rules are much more complex, you have to write much more ESQL code. ESQL provides many ways to access messages for validation, for example CAST statements can be used to check for correct date or time formats.

For validations against directory information, you can easily access any database table using standard message broker capabilities. For information in other directories, there are support packs that can be used, for example for information

in an LDAP directory you can use WebSphere BI MB support pack IA08. (You can find WebSphere BI MB support packs at http://www.ibm.com/software/integration/support/supportpacs.) Just in case you need to access other directories, you may have to write your own plug-in node that can access the information it that directory.

As long as the validations that you need to code yourself using such techniques are limited in the number of rules that you have to check and the number of different message types you have to validate is low, you would code them in ESQL. The SWIFT FIN standard, for example, knows more than 300 different message types and for most of them many different rules have to be checked. For this reason, WebSphere BI for FN has implemented its own message validation node. This node validates messages based on sets of rules. All rules apply always to a specific location in the message tree and are based on conditions.

To validate the message, the node traverses the complete message tree. This requires that a WebSphere BI MB parser can parse the complete message. In the case of messages that are not self-defining, a message set is required that describes the mapping from the physical stream to the logical structure. Using the parser, the message has been checked for conformity to the message structure. The WebSphere BI for FN validation engine checks at each element in the tree whether there is a validation rule for this element. A rule can validate a single element, or it can refer to elements that have to occur before or after the current element.

Even with this method, there are some drawbacks. To allow this kind of processing node and to define the validation rules, the complete message tree structure that is generated based on the message set must be known. Therefore, there is a tight coupling between these two usually independent parts. Another drawback with this method is that some errors are reported the WebSphere BI MB way using parser exceptions, while others are validation error codes conforming to the message standard.

In the validation node, the parser exceptions may be caught, but, in some situations, the information reported with the exception is not sufficient to map to the specific standard defined error code. In such cases, a mapping to a more general error code must be used, if possible. A third problem is that the parser stops parsing with the first parser problem. In this situation further errors cannot be detected.

If such disadvantages are not acceptable, there is still the possibility to parse the message yourself. There are two ways to do this. The first and easiest way is to write a plug-in node that parses and validates the bit stream. With this approach there have to be explicit invocations of this node in a message flow. Another, better but more complex, way would be writing your own validating message parser. Such a parser can parse and validate the message in many different ways. Whatever the requirement, validation can always be performed.

CONCLUSION

WebSphere BI MB already provides much functionality to validate messages that are processed in message flows. Messages can be checked for correct syntax and there are already many ways to check messages for correct content. Nevertheless, in many situations this is not sufficient. Depending on your requirements for message validation, you may need to do further checking. This has to be implemented by you, for example using a Compute node that validates dependencies across different fields in the message, checks values against the content of a database table, or just converts broker exceptions into error codes defined for the messages that you process. Other ways to implement message validation are by using plug-in nodes or plug-in parsers.

The capabilities of the message broker are steadily improving. This may reduce the effort that you have to make with your own programming.

Michael Groetzner
IBM (Germany)

Creating error dumps on Unix

INTRODUCTION

WebSphere MQ (WMQ) includes several features that discover problems and errors within applications using WMQ or within WMQ itself. There are three paths where error logs may be found, and it is possible to use **runmqsc** or other tools to check the WebSphere MQ configuration and functionality.

The error logging mechanism is very useful, but there are still situations where the level of knowledge of the WMQ administration people may not be sufficient to solve the problem. In such a case, help from the software vendor, IBM, is needed. The script described in this article is designed to collect all the data that is required by the IBM support staff. Following the naming convention on mainframes, I called this data collection 'error dump'.

THE ERROR DUMP SCRIPT

Aims for an error dump script

In the past, I have supported several hundred WebSphere MQ queue managers on different platforms – mostly Unix. In such a large environment there are always some queue managers that cause a lot of trouble with a high number of insoluble errors. In these cases I called IBM support to assist me in solving the problem. The IBM support staff need several items of information about the queue manager, as well as about the system where the queue manager resides.

To simplify and hasten the collection of the error dump data required by IBM, and in order to collect the complete data without missing any necessary files, I created this script. It may take a lot of time – and cost a lot of money – if IBM has to defer the problem solution because some error data is

missing. You will have to wait another week or two for the failure to occur again to give you your next chance to collect the missing data. Then perhaps on this occasion you will omit different data – and the problem solving is deferred again.

How the error dump script works

The script described here runs very quickly (it needs less than one minute). The script copies any error log file, FFST files, and any helpful system information to a temporary directory. Then the script counts and lists the whole process and thread list, any IPC segment such as shared memory or semaphores, and prints out information about the version of the operating system and so on. I know from experience that in nearly every situation this information satisfies the requirements of the IBM support staff.

The script collects some useful data that is available on any Unix system. On AIX and Sun Solaris systems, in addition to this general information, some error dump data is collected. If you use WebSphere MQ on other platforms, such as Linux or HP-UX, it is very easy to extend this script. The error dump data, specific to AIX and Sun Solaris, is collected and written within its own functions. Just create another function for your operating system and extend the case switch in line 351.

The options to display processes and threads using the tool **ps** differ between AIX and Sun Solaris, so the specific options are set in the operating system-specific functions. Nevertheless on Unix systems other than AIX or Sun Solaris at least a simple process list is created using the command **ps –ef**. This command is defined in the function *dump_data_other*.

Using the error dump script

An application failure that affects a WebSphere MQ queue manager, or a failure of the queue manager itself, requires information about the system during the failure. In such cases, the error dump script should be available on the machine. For my systems, I created a default environment containing

(among other things) this error dump script, so this script will always be available when an insoluble error occurs. Otherwise, you will need to copy this script to the machine when the failure occurs; this is possible, but, of course, takes up some precious minutes.

The script itself runs very quickly and normally takes less than one minute to collect all the required error dump data. Afterwards the system may be restarted (or whatever is needed to enable your business to run again). You have to open a problem record (PMR) at IBM and then send the created file to the IBM support address, referring to the PMR number. IBM will then have the necessary data and should be able to solve your problem.

If available, you should specify the name of the queue manager experiencing the problem when you run the error dump script. It is also possible to specify the parameter **--noqmgr** (written with two leading dashes) instead, but less information will be collected. For example the error files in */var/mqm/qmgrs/your_queuemanager/errors* (replace *your_queuemanager* with the real name of your queue manager experiencing problems) are not copied to the error dump file. But the attribute **--noqmgr** is useful when you have problems not specific to a queue manager, eg when you are not able even to create a queue manager.

SAMPLE ERROR DUMPS

This section lists the contents of the error dump files on AIX and Sun Solaris to show the differences between these two operating systems. Some files are the same (eg, the error log files) and some are similar (process and thread list, output of the top utility – if available). The other output files are completely different because of the system-specific commands, such as **lslpp** and **pkinfo**. Additional commands specific to AIX are **errpt**, **oslevel**, **lsdev**, **lsattr**, and **instfix**. Sun Solaris-specific commands are **sysdef**, **prtdiag**, **showrev**, and **prtconf**. On Sun Solaris the file */etc/system*, which is not available on AIX, is archived as well.

Listing of a sample error dump created on AIX with option --noqmgr

The following lines are created on an AIX system using the command **error_dump --noqmgr**:

```
-rw-r—r—    1 mqm mqm         74Ø 2ØØ5-Ø2-Ø7 Ø8:59 df-k.txt
-rw-r—r—    1 mqm mqm       26768 2ØØ5-Ø2-Ø7 Ø8:59 errpt-a.txt
-rw-r—r—    1 mqm mqm        928Ø 2ØØ5-Ø2-Ø7 Ø8:59 ipcs-a.txt
-rw-r—r—    1 mqm mqm        1566 2ØØ5-Ø2-Ø7 Ø8:59 lsattr-E-l_sysØ.txt
-rw-r—r—    1 mqm mqm        294Ø 2ØØ5-Ø2-Ø7 Ø8:59 lsdev-C.txt
-rw-r—r—    1 mqm mqm        1413 2ØØ5-Ø2-Ø7 Ø8:59 lslpp-h.txt
-rw-r—r—    1 mqm mqm       19917 2ØØ5-Ø2-Ø7 Ø8:59 ls-lR.txt
-rw-r—r—    1 mqm mqm           8 2ØØ5-Ø2-Ø7 Ø8:59 oslevel.txt
-rw-r—r—    1 mqm mqm       18Ø56 2ØØ5-Ø2-Ø7 Ø8:59 ps-efmo.txt
-rw-r—r—    1 mqm mqm          29 2ØØ5-Ø2-Ø7 Ø8:59 uname-a.txt
-rw-r—r—    1 mqm mqm       92ØØ5 2ØØ5-Ø2-Ø7 Ø9:ØØ instfix-i.txt

var:
total Ø
drwxr-xr-x    4 mqm mqm         12Ø 2ØØ5-Ø2-Ø7 Ø9:ØØ mqm

var/mqm:
total 5
-rw-r—r—    1 mqm mqm        2222 2ØØ5-Ø2-Ø7 Ø8:57 mqs.ini
drwxr-xr-x    2 mqm mqm         688 2ØØ5-Ø2-Ø7 Ø8:59 errors
drwxr-xr-x    3 mqm mqm          72 2ØØ5-Ø2-Ø7 Ø9:ØØ qmgrs

var/mqm/errors:
total 1Ø6Ø
-rw-r—r—    1 mqm mqm           Ø 2ØØ4-Ø9-1Ø 1Ø:59 AMQERRØ2.LOG
-rw-r—r—    1 mqm mqm           Ø 2ØØ4-Ø9-1Ø 1Ø:59 AMQERRØ3.LOG
-rw-r—r—    1 mqm mqm       28361 2ØØ4-Ø9-1Ø 1Ø:59 AMQ14372.Ø.FDC
-rw-r—r—    1 mqm mqm       19870 2ØØ4-Ø9-13 18:16 AMQ14198.Ø.FDC
-rw-r—r—    1 mqm mqm       20571 2ØØ4-1Ø-Ø1 17:Ø4 AMQ12318.Ø.FDC
-rw-r—r—    1 mqm mqm      168Ø95 2ØØ5-Ø2-Ø7 Ø8:59 AMQØ2286.Ø.FDC
-rw-r—r—    1 mqm mqm       53Ø08 2ØØ5-Ø2-Ø7 Ø8:59 AMQ13Ø06.Ø.FDC
-rw-r—r—    1 mqm mqm       63Ø65 2ØØ5-Ø2-Ø7 Ø8:59 AMQ13944.Ø.FDC
-rw-r—r—    1 mqm mqm       64Ø02 2ØØ5-Ø2-Ø7 Ø8:59 AMQ17868.Ø.FDC
-rw-r—r—    1 mqm mqm       37531 2ØØ5-Ø2-Ø7 Ø8:59 AMQ18Ø1Ø.Ø.FDC
-rw-r—r—    1 mqm mqm       21Ø92 2ØØ5-Ø2-Ø7 Ø8:59 AMQ19Ø98.Ø.FDC
-rw-r—r—    1 mqm mqm       54392 2ØØ5-Ø2-Ø7 Ø8:59 AMQ19352.Ø.FDC
-rw-r—r—    1 mqm mqm       5Ø253 2ØØ5-Ø2-Ø7 Ø8:59 AMQERRØ1.LOG

var/mqm/qmgrs:
total Ø
drwxr-xr-x    3 mqm mqm          72 2ØØ5-Ø2-Ø7 Ø9:ØØ @SYSTEM

var/mqm/qmgrs/@SYSTEM:
total Ø
```

```
drwxr-xr-x    2 mqm mqm          48 2004-09-09 15:42 errors

var/mqm/qmgrs/@SYSTEM/errors:
total 0
```

Listing of a sample error dump created on Sun Solaris with option --noqmgr

The following lines are created on a Sun Solaris system using the command **error_dump --noqmgr**:

```
-rw-r--r--    1 mqm mqm         768 2005-02-07 08:12 df-k.txt
-rw-r--r--    1 mqm mqm        2804 2005-02-07 08:12 etc_system.txt
-rw-r--r--    1 mqm mqm        9314 2005-02-07 08:12 ipcs-a.txt
-rw-r--r--    1 mqm mqm       27306 2005-02-07 08:12 ls-lR.txt
-rw-r--r--    1 mqm mqm         530 2005-02-07 08:12 pkginfo-l.txt
-rw-r--r--    1 mqm mqm        3133 2005-02-07 08:12 prtconf.txt
-rw-r--r--    1 mqm mqm        1302 2005-02-07 08:12 prtdiag-v.txt
-rw-r--r--    1 mqm mqm       30740 2005-02-07 08:12 ps-efPL.txt
-rw-r--r--    1 mqm mqm       24433 2005-02-07 08:12 showrev-a.txt
-rw-r--r--    1 mqm mqm       13194 2005-02-07 08:12 sysdef-i.txt
-rw-r--r--    1 mqm mqm          61 2005-02-07 08:12 uname-a.txt

var:
total 0
drwxr-xr-x    4 mqm mqm         120 2005-02-07 08:12 mqm

var/mqm:
total 5
-rw-r--r--    1 mqm mqm        2275 2005-02-07 08:10 mqs.ini
drwxr-xr-x    2 mqm mqm        1072 2005-02-07 08:12 errors
drwxr-xr-x    3 mqm mqm          72 2005-02-07 08:12 qmgrs

var/mqm/errors:
total 5122
-rw-r--r--    1 mqm mqm      256578 2003-12-02 10:48 AMQERR03.LOG
-rw-r--r--    1 mqm mqm       25972 2005-01-19 09:50 AMQ08915.0.FDC
-rw-r--r--    1 mqm mqm       10225 2005-01-20 21:00 AMQ23003.0.FDC
-rw-r--r--    1 mqm mqm      256210 2005-02-03 11:48 AMQERR02.LOG
-rw-r--r--    1 mqm mqm       43723 2005-02-07 08:12 AMQ26080.0.FDC
-rw-r--r--    1 mqm mqm       28289 2005-02-07 08:12 AMQ26081.0.FDC
-rw-r--r--    1 mqm mqm       19178 2005-02-07 08:12 AMQ26082.0.FDC
-rw-r--r--    1 mqm mqm       25461 2005-02-07 08:12 AMQ26083.0.FDC
-rw-r--r--    1 mqm mqm       20856 2005-02-07 08:12 AMQ26084.0.FDC
-rw-r--r--    1 mqm mqm       20349 2005-02-07 08:12 AMQ26085.0.FDC
-rw-r--r--    1 mqm mqm      104759 2005-02-07 08:12 AMQ26087.0.FDC
-rw-r--r--    1 mqm mqm       64952 2005-02-07 08:12 AMQERR01.LOG

var/mqm/qmgrs:
```

```
total 0
drwxr-xr-x    3 mqm mqm          72 2005-02-07 08:12 @SYSTEM

var/mqm/qmgrs/@SYSTEM:
total 0
drwxr-xr-x    2 mqm mqm          48 2003-08-22 12:24 errors

var/mqm/qmgrs/@SYSTEM/errors:
total 0
```

Listing of a sample error dump with queue manager specified

The output of the command is nearly the same when a valid and running queue manager is specified. In addition to the samples above, some files specific to the queue manager (named TEST in this sample) are listed too:

```
...

var/mqm/qmgrs:
total 0
drwxr-xr-x    3 mqm mqm          72 2005-02-07 08:12 @SYSTEM
drwxr-xr-x    3 mqm mqm          96 2005-02-07 08:12 TEST

...

var/mqm/qmgrs/TEST:
total 4
drwxr-xr-x    2 mqm mqm         144 2005-02-07 08:10 errors
-rw-r—r—    1 mqm mqm        1253 2005-02-07 08:10 qm.ini

var/mqm/qmgrs/TEST/errors:
total 4
-rw-r—r—    1 mqm mqm        1218 2005-02-07 08:10 AMQERR01.LOG
-rw-r—r—    1 mqm mqm           0 2005-02-07 08:10 AMQERR02.LOG
-rw-r—r—    1 mqm mqm           0 2005-02-07 08:10 AMQERR03.LOG
```

DESCRIPTION OF THE CODE

The script runs in several steps. First the function *initialise* checks whether the queue manager specified as a parameter on the command line exists. This check happens in the function *check_qmgr*. If the queue manager is not available, a usage message is shown. Otherwise the function checks the data path if it is not */var/mqm/qmgrs/name_of_the_queuemanager* and defines a parameter

containing the name of this data path. Then the script extends the program search path and looks for the **top** utility. If present, the path to this tool is stored in the parameter TOP.

The next step of the script is to collect some operating system-independent data. Within the function *dump_data_all*, first the signal USR2 is sent – using the kill command – to all processes containing the string *amq*, as well as the name of the troublesome queue manager. This signal ensures that the processes dump their actual state in the form of FFST files. The processes will then continue running.

If your operating system is AIX or Sun Solaris, the script will collect some further error dump data, such as processes and threads, version of the operating system, device configuration, and installed WebSphere MQ version and patches. If you use additional Unix versions such as HP-UX or Linux, you may create similar functions and add the case value in line 351 of the script. Use only lower case letters (eg linux instead of Linux), because the parameter $OS, which is filled in the function *initialise*, contains the output of the command **uname -s**, but translated with the tool **tr** to lower case characters only.

Next some files (error files, ini files) are copied to the temporary directory within the function *copy_files*. Then the error dump data is collected and stored in one single output file using the **tar** program. The tar file is compressed afterwards, to save disk space. The error dump file may now be sent to the IBM support staff.

LISTING OF THE ERROR DUMP SCRIPT

```
 1 #!/bin/ksh
 2 #
 3 ##################################################################
 4 #
 5 #   Script to create an error dump for WebSphere MQ.
 6 #
 7 ##################################################################
 8 #
 9 #   DESCRIPTION
10 #
```

```
11 #   This script creates a WebSphere MQ error dump, to
12 #   open a problem record at IBM. The script collects
13 #   all relevant data, which IBM needs for input.
14 #
15 #   May, 13th 2004
16 #   Hubert Kleinmanns (N-Tuition AG)
17 #
18 ###################################################################
19
20 #
21 # Show a usage message.
22 #
23
24 function show_usage
25 {
26    ret=$1
27
28    # Display specific error message.
29    case $ret in
30       1)
31          echo "Invalid number of parameters..."
32          ;;
33       2)
34          echo "Parameter \"$2\" invalid..."
35          ;;
36       3)
37          echo "Queue manager \"$2\" not found..."
38          ;;
39    esac
40
41    # Specify a queue manager, if possible.
42    echo "usage\t`basename $PROG_NAME` queuemanager"
43    echo "usage\t`basename $PROG_NAME` –noqmgr"
44
45    exit $ret
46 }
47
48 #
49 # Check whether a specified queue manager exists and store the
50 # prefix of this queue manager.
51 #
52
53 function check_qmgr
54 {
55    # Initialize the data path of the queue manager.
56    QMGR_DATA_PATH=""
57
58    # Initialize some local parameters.
59    name=""
60    prefix=""
```

```
61    dir=""
62    new_stanza="n"
63
64    # Read all lines of the file 'mqs.ini'.
65    cat /var/mqm/mqs.ini | while read line
66    do
67       # The stanza 'QueueManager' was found.
68       if [ "$line" = "QueueManager:" ]
69       then
70          # Mark the beginning of the stanza.
71          new_stanza="y"
72
73          # Read next line.
74          continue
75       fi
76
77       # Read the values of this stanza.
78       if [ "$new_stanza" = "y" -a "X$line" != "" ]
79       then
80          # Read the label of the stanza attribute.
81          label='echo "$line" | awk -F'=' '{print $1}''
82
83          # Read the value of the stanza attribute.
84          value='echo "$line" | awk -F'=' '{print $2}''
85
86          # Store the stanza attribute.
87          case $label in
88             # Attribute is the queue manager name.
89             Name)
90                name="$value"
91                ;;
92
93             # Attribute is the queue manager prefix.
94             Prefix)
95                prefix="$value"
96                ;;
97
98             # Attribute is the queue manager directory.
99             Directory)
100               dir="$value"
101               ;;
102         esac
103
104       # Check whether all three stanza attributes have been filled.
105         if [ "X$name" != "X" -a "X$prefix" != "X" -a "X$dir" != "X"
]
106         then
107   # Check whether the stanza belongs to the specified queue manager.
108            if [ "$name" = "$1" ]
109            then
```

```
110                # Store the queue manager name in a global parameter.
111                QMGR_NAME="$name"
112
113                # Store the data path of the queue manager.
114                QMGR_DATA_PATH="$prefix/qmgrs/$dir"
115
116                # Return successfully.
117                return 0
118             else
119                # Initialize the local parameters for the next
stanza.
120                name=""
121                prefix=""
122                dir=""
123
124                # Read next line.
125                continue
126             fi
127          fi
128       fi
129    done
130
131    # Return with error.
132    return 3
133 }
134
135 #
136 # Initialize the script parameters.
137 #
138
139 function initialise
140 {
141    # If no qmgr is specified set dummy value.
142    case $1 in
143       –noqmgr)
144          QMGR_NAME="noqmgr"
145          ;;
146       -*)
147          show_usage 2 $1
148          ;;
149       *)
150          # Check whether the specified queue manager exists.
151          check_qmgr $1
152          ret=$?
153
154          [ $ret -ne 0 ] && show_usage $ret $1
155          ;;
156    esac
157
158    # Get the operating system.
```

```
159    OS='uname -s | tr '[:upper:]' '[:lower:]''
160
161    # Define some directories, which have to be included in the
search path.
162    REQ_DIRS="/usr/bin /usr/sbin /etc /usr/local/bin"
163
164    # Store the actual path and separate the directories.
165    path='echo $PATH | tr ':' ' ' | sed -e "s/^/ /" | sed -e "s/$/ /
"'
166
167    # Check, whether the directories defined above are already part
of the path ...
168    for dir in $REQ_DIRS
169    do
170       chk='echo "$path" | grep " $dir "'
171
172       # ... append the directory to the path otherwise.
173       [ "X$chk" = "X" ] && PATH=$PATH:$dir
174    done
175
176    # Find the 'top' utility.
177    TOP='which top'
178    [ -e "$TOP" ] || TOP=""
179
180    # Set the output file name.
181    OUT_FILE=$HOME/${QMGR_NAME}_'date +"%Y%d%m-%H%M"'.tar
182
183    # Specify the root directory for storing the dump data.
184    ERRDUMP="$TEMP_DIR/errdump"
185
186    # Create the dump directory.
187    mkdir -p $ERRDUMP
188 }
189
190 #
191 # Function to dump data, which is relevant on all operating systems.
192 #
193
194 function dump_data_all
195 {
196    # Send the signal USR2 to all MQ processes. The processes create
an FDC file
197    # in /var/mqm/errors and dump their actual state. Then the
processes continue.
198    if [ "$QMGR_NAME" != "noqmgr" ]
199    then
200       proc_list='ps -ef | grep amq | grep $QMGR_NAME | awk '{print
$2}''
201
202       [ "X$proc_list" != "X" ] && kill -USR2 $proc_list
```

```
203    fi
204
205    # Store some system information.
206    uname -a > $ERRDUMP/uname-a.txt
207
208    # List all available IPC segments.
209    ipcs -a > $ERRDUMP/ipcs-a.txt
210
211    # List all files in the MQ data and ignore duplicate files.
212    ls -lR /var/mqm $QMGR_DATA_PATH | sort -u > $ERRDUMP/ls-lR.txt
2>/dev/null
213
214    # Display the disk configuration.
215    df -k > $ERRDUMP/df-k.txt
216 }
217
218 #
219 # Function, which collects data using AIX-specific tools.
220 #
221
222 function dump_data_aix
223 {
224    # Count all active processes.
225    echo "`ps -ef | wc -l` processes found" > $ERRDUMP/ps-efmo.txt
226
227    # Count all active threads.
228    echo "`ps -efmo THREAD | wc -l` threads found\n" >> $ERRDUMP/ps-
efmo.txt
229
230    # List all active processes and threads.
231    ps -efmo THREAD >> $ERRDUMP/ps-efmo.txt
232
233    # If available, run the 'top' command.
234    [ "$TOP" != "" ] && $TOP -Count 1 > $ERRDUMP/top-Count1.txt
235
236    # Store the level of the operating system.
237    oslevel > $ERRDUMP/oslevel.txt
238
239    # List all known devices.
240    lsdev -C > $ERRDUMP/lsdev-C.txt
241
242    # List some system attributes.
243    lsattr -E -l sys0 > $ERRDUMP/lsattr-E-l_sys0.txt
244
245    # Display installed MQ software packages.
246    lslpp -h "mq*" > $ERRDUMP/lslpp-h.txt
247
248    # Display messages created by the error reporter tool.
249    errpt -a > $ERRDUMP/errpt-a.txt
250
```

```
251    # Show the installed fix level.
252    instfix -i > $ERRDUMP/instfix-i.txt
253 }
254
255 #
256 # Function, which collects data using Sun Solaris-specific tools.
257 #
258
259 function dump_data_sunos
260 {
261    # Count all active processes.
262    echo "`ps -efP | wc -l` processes found" > $ERRDUMP/ps-efPL.txt
263
264    # Count all active threads.
265    echo "`ps -efPL | wc -l` threads found\n" >> $ERRDUMP/ps-efPL.txt
266
267    # List all active processes and threads.
268    ps -efPL >> $ERRDUMP/ps-efPL.txt
269
270    # If available, run the 'top' command.
271    [ "$TOP" != "" ] && $TOP -d 1 -n 400 > $ERRDUMP/top-d1-n400.txt
272
273    # List some system attributes.
274    sysdef -i > $ERRDUMP/sysdef-i.txt
275
276    # List all known devices.
277    /usr/platform/sun4u*/sbin/prtdiag -v > $ERRDUMP/prtdiag-v.txt
278
279    # Show the installed system patches.
280    showrev -a > $ERRDUMP/showrev-a.txt
281
282    # List the system configuration.
283    prtconf > $ERRDUMP/prtconf.txt
284
285    # Copy the file '/etc/system'.
286    cp /etc/system $ERRDUMP/etc_system.txt
287
288    # Display installed MQ software packages.
289    pkginfo -l 'pkginfo | grep mq | awk '{print $2}'' > $ERRDUMP/
pkginfo-l.txt
290 }
291
292 #
293 # Function, which creates a system-independent process list.
294 #
295
296 function dump_data_other
297 {
298    # Count all active processes.
299    echo "`ps -ef | wc -l` processes found" > $ERRDUMP/ps-ef.txt
```

```
300
301    # List all active processes.
302    ps -ef >> $ERRDUMP/ps-ef.txt
303 }
304
305 #
306 # Copy relevant MQ files to the error dump path.
307 #
308
309 function copy_files
310 {
311    # Copy queue manager independent files.
312    mkdir -p $ERRDUMP/var/mqm/qmgrs/@SYSTEM
313    cp -rp /var/mqm/mqs.ini /var/mqm/errors $ERRDUMP/var/mqm
314    cp -rp /var/mqm/qmgrs/@SYSTEM/errors $ERRDUMP/var/mqm/qmgrs/
@SYSTEM
315
316    # Copy the files, which belong to the specified queue manager.
317    if [ "$QMGR_NAME" != "noqmgr" ]
318    then
319       mkdir -p $ERRDUMP$QMGR_DATA_PATH
320       cp -rp $QMGR_DATA_PATH/errors $QMGR_DATA_PATH/qm.ini
$ERRDUMP$QMGR_DATA_PATH
321    fi
322 }
323
324 ##########################################################################
325 #
326 #   MAIN function.
327 #
328 ##########################################################################
329
330 # Define a directory to store temporary files.
331 TEMP_DIR="/tmp"
332
333 # Store the name of the script.
334 PROG_NAME=$0
335
336 # Invalid number of parameters.
337 [ $# -ne 1 ] && show_usage 1
338
339 # Run the initialization function.
340 initialise $*
341
342 # Change to the standard MQ data path.
343 cd $TEMP_DIR
344
345 # Dump OS independent data.
346 dump_data_all
347
```

```
348 # Dump data specific to the operating systems AIX and Sun Solaris.
349 case $OS in
350    aix|sunos)
351       dump_data_$OS
352       ;;
353
354    *)
355       dump_data_other
356       ;;
357 esac
358
359 # Copy some files to the temporary directory.
360 copy_files
361
362 # Archive the stored files.
363 tar cvf $OUT_FILE errdump
364
365 # Compress the output file.
366 compress $OUT_FILE
367
368 cd -
369
370 # Clean-up the dump directory.
371 rm -rf $ERRDUMP
372
373 exit 0
374
375 ################################################################
376 #
377 #   End of error dump script.
378 #
379 ################################################################
```

Hubert Kleinmanns
Senior Consultant
N-Tuition Business Solutions (Germany)

MQSeries checklist

Our shop runs MQSeries for VSE Version 2 Release 1.2. We have VSE/ESA Version 2 Release 5, with CICS/VSE Version 2 Release 3 and VTAM for VSE/ESA Version 4 Release 2. MQSeries has been running stably on our machines for over half a decade, but recently we noticed that it was becoming

unstable. As with any system that was once stable but is no longer, we looked for what had changed. The checklist that we followed is shown below.

CHANGES TO HARDWARE

Our first check was to see if there were any changes to our hardware. The reason for checking is that sometimes the installation of new hardware can cause changes to the timing of some application programs. Obviously, if your application system is reliant on a processing sequence and that sequence changes there could be problems. In our shop there were no hardware upgrades that were relevant.

SOFTWARE CHANGES

The installation of new software or software updates is probably the biggest reason for problems with application systems. Obviously, make sure that no software has been downloaded from the Internet onto the live production system. Software should be run and checked on your test system first. PTFs can be a big source of problems, one of the biggest issues being their incorrect application. Usually, applying a service is straightforward; however, sometimes IBM hides additional instructions in the PTF service cover letter, or even in the JCL of the PTF. These can be quite important, and problems will occur if they are not carried out. Examples include:

- *Modify CSD* – applying a PTF can result in changes to the CICS CSD. This includes programs, files, or changes to entries. You should apply the modifications provided in MQJCSD.Z or application instability could result.
- *MQJSETUP.Z and MQSU* – PTFs can sometimes include a change to the configuration file in MQSeries. This is found in SYSIN.Z. Along with any updates, the file is reloaded by the MQJSETUP.Z job. Once loaded, the MQSU transaction applies the new version of the file to the MQSeries configuration file. It is evident that if these

instructions are not followed, the configuration can be out of sync and this can be a cause of system instability. One way to check this is to see whether the number of records loaded (which is displayed when the MQSU transaction is completed) corresponds to the number printed in the job output of MQJSETUP.

- *Modify DCT* – there are some occasions when installing a PTF can cause changes to MQSeries DCT entries or new entries. These can be found in part MQCICDCT.Z. If the CICS DCT is not rebuilt to incorporate the changes when instructed, instability could ensue.

KNOWN MQSERIES ISSUES

IBM has a list of known problems with MQSeries for VSE. These include:

- *Abends with AFCL, AFCP, or AJCN during execution of MQSeries*. This is a well-known, widely-reported problem that results from configuration problems with the CICS system journal (the CICS system journal needs to be properly defined using buffer values that are applicable to the biggest VSAM record that MQSeries will access).
- *TP names in channel definitions*. The MQSeries for VSE sender channel definitions created by MQMT option 1.3 require a reference to a TP Name field. This TP Name field needs to be precisely defined. This is very important when a partner system is located on a Windows, OS/2, or Unix system, because it could be case sensitive. The profile definition on the partner needs to reference a valid program name in an actual directory.
- *Starting and stopping TCP/IP*. Shutting down TCP/IP with MQSeries active will cause the TCP/IP Listener transaction (MQTL) and any active TCP/IP channels to abend. In reverse, if TCP/IP is not active when MQSeries is started, the TCP/IP Listener program will not start. This is because the TCP/IP socket services are stopped when TCP/IP is

shut down. In order to restart the TCP/IP channels both TCP/IP and the Listener program need to be restarted.

WORKLOAD CHANGES

In our shop it was found that there had been a significant increase in workload. A major new project had gone live on a partner system and this was transmitting to our MQSeries for VSE system, which was made evident by increases in network traffic. Furthermore, the system was catering for the international market, which meant that there were increases in the overnight workloads, and some new data types had been introduced. In our case it was necessary to increase our disk and processor resources to cater for the increased workload.

CONCLUSION

The overall conclusion with problem determination in systems that have been stable and begin to exhibit signs of instability is that you must determine what has changed. Following the simple categories above is a useful way of getting to the root of the problem. Check the software, hardware, known problems, and workload to determine where the problem is.

John Edwards
Systems Administrator (UK)

Fiorano Software has announced Version 8.0 of FioranoMQ, its stand-alone Java Message Service (JMS) server.

The product delivers enhancements for enterprise-grade messaging in a number of key areas, from improved enterprise management capabilities and new administrative/ configuration tools to a new Component Assembly Framework that allows for more efficient in-process message routing and dispatching.

For further information contact:
URL: www.fiorano.com/downloads/fmq/fmqreleasenotes.pdf.

* * *

Capitalware has announced Version 1.3 of MQ Visual Edit, its tool for letting developers and administrators view, manipulate, and manage messages in a WebSphere MQ queue and presenting the data in a simplified format similar to a database utility or spreadsheet program.

Version 1.3 of the Java-based tool offers a number of user-interface enhancements and bug fixes, including a single shell script for running it on Linux, Mac OS X, and Unix.

MQ Visual Edit can be used by application programmers, JMS developers, quality assurance testers, and production support personnel. The tool allows for quick problem solving because the data is presented in a very logical and insightful manner, they claim.

MQ Visual Edit can run on any platform that supports Java Version 1.3 (or higher).

For further information contact:
URL: www.capitalware.biz/products.html#mqve.

* * *

IBM has announced WebSphere Studio Asset Analyzer, which helps IT executives analyse and understand their existing application code.

The company has also announced Version 5.1.2 of WebSphere Enterprise Developer (WSED), which contains enhancements that are intended to make Web, traditional, and integrated development faster, and developer communities more productive. WSED consists of a common workbench, and integrated set of tools that support end-to-end, model-based, application development, run-time testing, and rapid deployment of e-business applications. WSED Version 5.1.2 supports new industry standards that simplify the development of Web user interfaces, and business logic. It includes productivity tools for business-oriented developers, new to Java, that integrate zSeries COBOL and PL/I processing via Web Services.

For further information contact your local IBM representative.

* * *