IBM Software

Impact2010

The Premier Conference for Business and IT Leaders

WebSphere Message Broker: Designing for Performance (1346A)

Andy Piper

Discover. Interact. Optimize.

MAY 2–7 Las Vegas, NV

Designing for Performance

- Highlighting major design issues which should be addressed when implementing a message processing solution using WebSphere Message Broker.
- Detail of how to minimize the costs of processing a message in order to produce more efficient message flows.
- Overview of the performance improvements that have been delivered recently

Overview

N O T E S

- The purpose of this presentation is to highlight the major design issues which should be addressed when implementing a message processing solution using WebSphere Message Broker (Message Broker).
- The level of performance which is obtained with a message flow is a function of the efficiency of the message flow and the environment in which the message flows runs. Well performing systems rarely happen by accident and must be planned. This presentation discusses those aspects of message flow design and broker configuration which you must focus on in order to produce an implementation which performs well.
- The issues raised in this presentation are cross platform and not dependent on any particular platform. The function discussed in the presentation is available on all of the platforms on which brokers are implemented.

What can you hope to get out of this presentation ?

- You will hopefully understand the component costs of processing messages in a Message Broker message flow and understand how to minimize the costs of processing in order to produce more efficient message flows. The benefit of doing this is that you can use less resource to achieve a given message rate or achieve a higher message rate with a given level of resource.
- It is possible to process messages with Message Broker in a variety of ways by structuring data or message flows differently. You could for example have a single message flow to process all messages. This would have different processing characteristics from a flow which processed only one of the message formats. In order to produce message flows which perform well it is important to invest time and effort thinking about how you will address issues such as data format and structure, message flow properties and error processing. In this presentation we are going to look at the issues which you need to address and discuss some recommendations for message and message flow design.
- In implementing a brokering solution it is important to consider all aspects of the system. Without sufficient CPU even the most efficient of message flows will not be able to process messages at the required rate.
- An understanding of the performance improvements that have been delivered with Message Broker V6.1

Contents

Contents

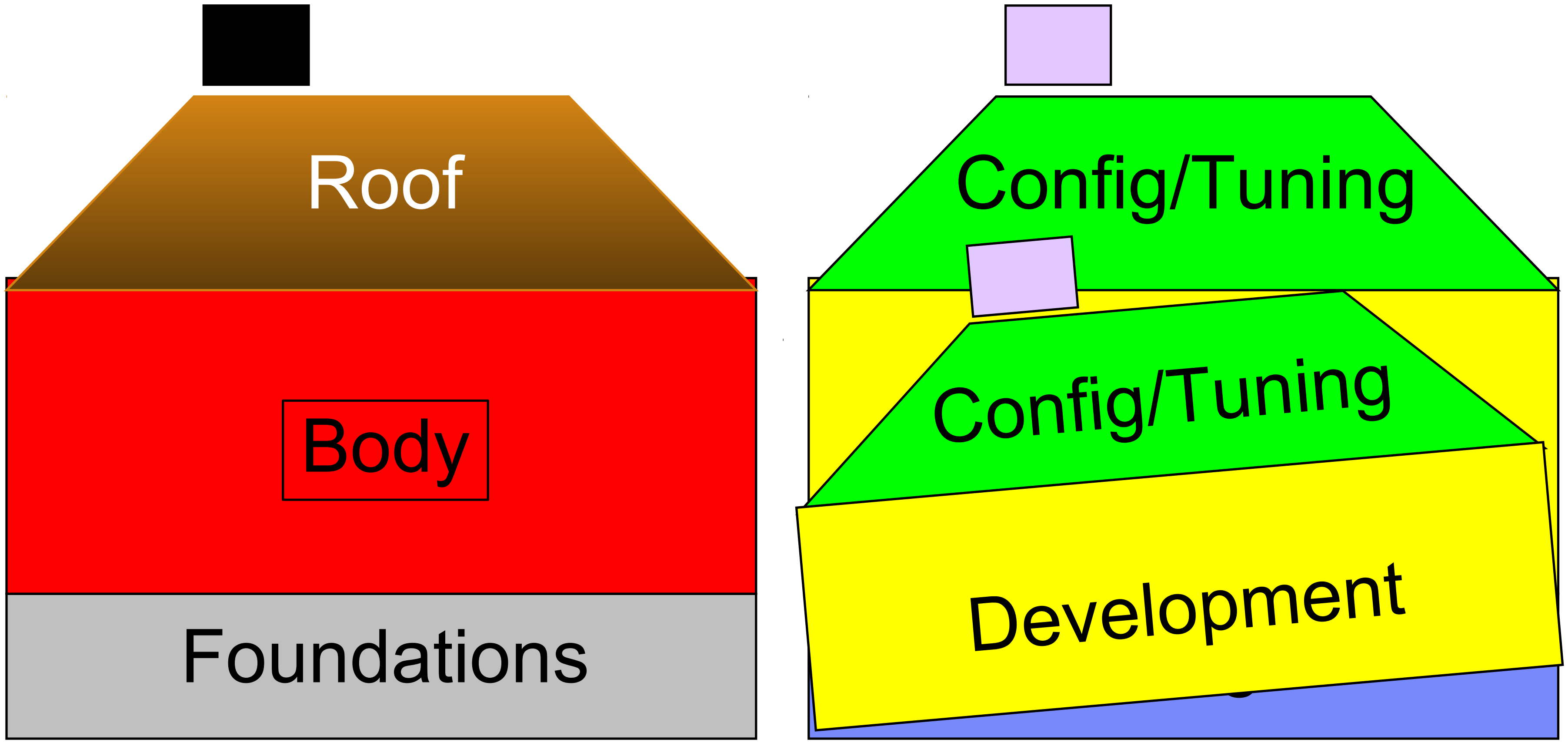
N O T E S

- In this presentation we will look at a variety of topics all of which touch on performance.

- In the Introduction we will quickly discuss the need to include detailed solution design at an early stage of the project. We also highlight on the need to cover many components of the solution, ranging from the applications which are being connected all the way through to the hardware on which the message flows will run. In this section we will also touch on understanding the requirements of any particular integration project since these will significantly affect the final outcome. These requirements are the input to you design process.

- When designing a message flow you need to consider a number of topics before finalising a design in order to ensure that you use Message Broker most efficiently:
  - The processing which needs to be performed in order to populate, manipulate and serialise the message tree.
  - The performance and reliability characteristics of the transport that is used for the input and output of data.
  - The structure of the message flow. This is similar to program structure.
  - Which transformation technology to choose and what the consequences of mixing technologies are.
  - The need for transactional processing and what the implications of using it are.
  - Techniques to cope with the serialisation of message processing.
- Tuning. We will discuss the need for tuning in the different areas.
- There are a number of problems which we see repeated as people use Message Broker. In this section we will highlight the most common problems in the hope that you can now avoid them.
- The section on Tools will highlight three different tools which can be used to help evaluate the performance of your message flows. Two are concerned with message generation. The third gives you the ability to control message broker (start, stop components etc.) and look at the accounting and statistics data which Message Broker is capable of producing.
- We will have a quick look at the key performance improvements that were introduced with Message Broker V6.1.
- Finally we will finish with a summary.

The Key Building Blocks

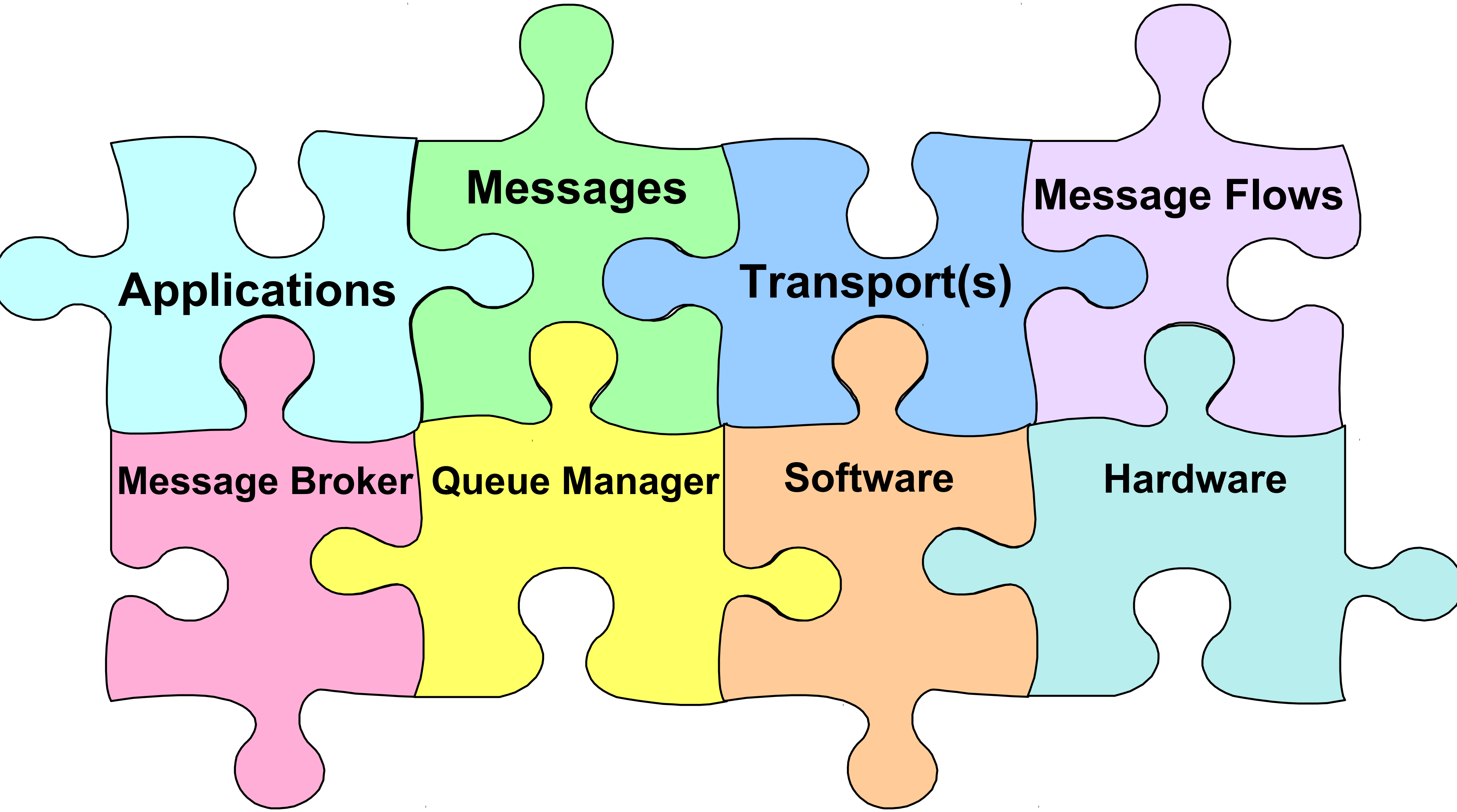


The key building blocks

N O T E S

- ❑ When we start to construct anything it is important that we do things in the correct sequence for the best possible outcome. Take the example of an office or house. When we start it is important to get solid foundations. Otherwise the house is danger of falling down. With the foundations in place we then build the body of the house before then moving on to add the roof.
- ❑ We can apply this analogy to integration projects using Message Broker.
- ❑ In this case our foundation is the design, not just of the message flow but of the whole system – that is the ways in which applications interact, the processing models that are used to achieve this, the system on which Message Broker runs and so on.
- ❑ The next step is the development of the message flows and the runtime environment. This is the implementation of the design. This is also an important stage to get correct (as they all are of course). Poor or inefficient coding can be the cause of many performance problems. Later in the presentation there are a number of tips to help you avoid the most common errors.
- ❑ Once we have a running system we can then apply configuration and tuning. Tuning only works when we have an implementation in place. Tuning is an optimisation process. We should not rely on it to turnaround a poorly performing situation. We cannot tune our way out of poor design or poorly written code. That is we see configuration and tuning as the top part of the house.
- ❑ Design is very important. If you produce a weak design you have the equivalent of subsidence. When you have that you can expect the building, or in this case performance of the ESB to collapse. So remember design is a vital process.

Assembling the Pieces



Introduction

N
O
T
E
S

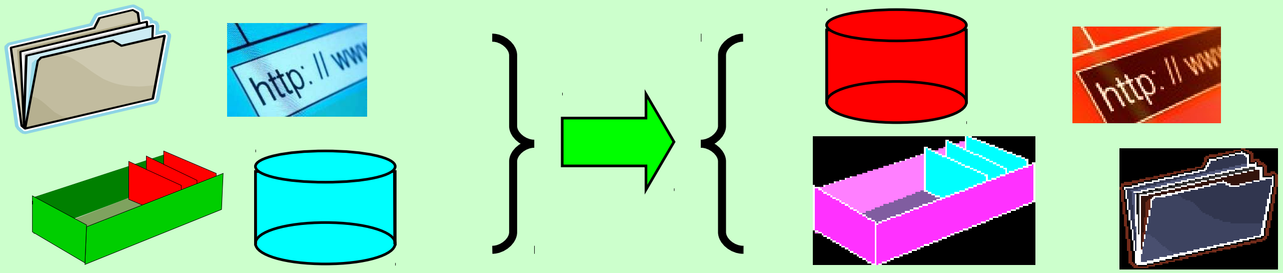
- It is important to look at all aspects of Message Broker implementation. These include
  - The ways in which the external applications will interact
  - The messages that will be processed
  - The Message Flows
  - The Configuration of the Message Broker
  - The Message Broker queue manager and any other queue managers which form part of the configuration
  - The transports over which messages are carried
  - The high level characteristics of the hardware and software which you choose to run the message flows on.
  - The type and characteristics of the hardware on which Message Broker runs.

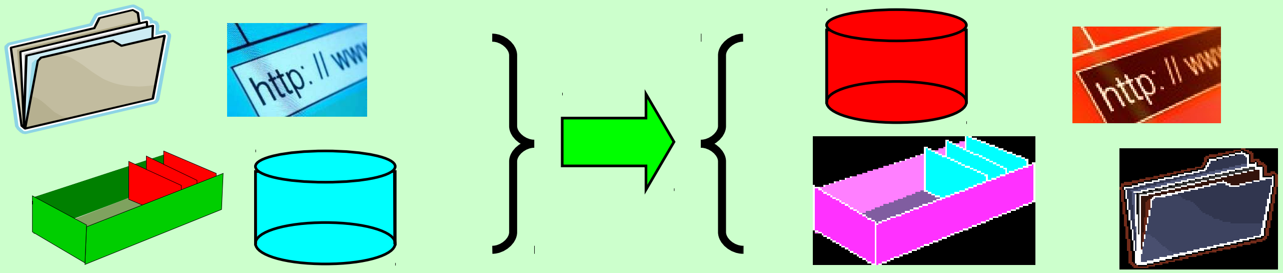
  These all play in part in overall performance and as such they should all be considered from day 1.

- It helps in planning your message flow design if are able to understand whether processing is likely to be CPU or I/O bound. This is particularly important when you need to achieve high volumes of processing and so achieve higher CPU utilisations. You do not want to be I/O bound and limited to 10% CPU utilisation for example.

- Understanding what Message Broker is, and perhaps more importantly what it is not, will help significantly in ensuring that you build an efficient application. As it says on the can it is a broker. It is not an application server. It is not a place to hold state. Understand the ingredients, follow the recommended cooking instructions (uses, design and coding guidelines) and you will be well on the way to building an efficient implementation that performs well.

Processing Requirements

Data	XML CSV SWIFT IDOC Bytes…MBytes C Struct COBOL copybook
Transport	
Processing Style/pattern	Transformation Routing Request/Reply Aggregation Warehousing
QOS	Best efforts vs assured Clustering Failover Persistent vs non persistent



Processing Requirements

N
O
T
E
S

- It is important to recognise what you want message broker to do for you in any given situation. There are a number of different aspects of processing that have to be considered:
  - What type of date will be processed, what format will it have
  - Size. We cannot process a message which is 100 MB at the same speed as one which is 100 bytes.
  - Transport. Some transports are more reliable than others. TCP/IP is an unreliable transport. In some cases a message flow might be used to save a copy of the incoming message to an MQ queue or a database. This changes the nature of the processing. Disk performance might become the limiting factor.
  - Processing model. Dependent on how we want message flows to interact we might want some coordination between a request and a reply. This can be done but comes at a cost. We will look at some of the common processing models.
  - Quality of service (QOS). What are your requirements for transactional processing, assured delivery, availability, restart etc. These non-functional requirements can and do have a significant impact on performance. Providing such facilities may well impact detailed design. It will certainly impact development, configuration and tuning.

- You may well have different requirements for different projects. When those different projects run in the same infrastructure then you have to think carefully about what facilities that infrastructure provides and also ensure that one project does not impact another, that is they do not have mutually exclusive requirements. If that is the case you may need to provide more than message broker.

Requirements

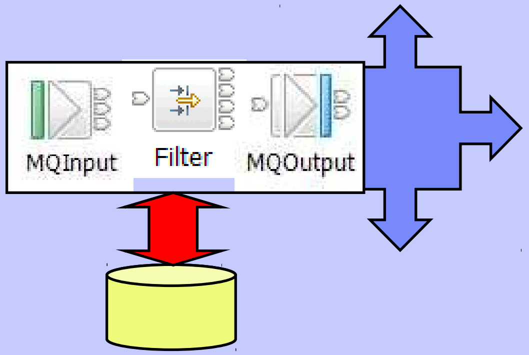
NOTES

- It is important to bound the processing that you will do with Message Broker. In a typical IT environment there are many interacting components, each with a role to play. It is important that you are able to clearly define the role of a message broker in the new environment. It can be easy to implement function which runs in the broker but which is not appropriate. This happens sometimes unfortunately. It is made all the more easier because there are a variety of transformation technologies and several programming languages at your disposal.
- A key design decision is deciding how much business logic to implement within your message flows. Given the capabilities of Message Broker there is the potential to implement large amounts of processing within a message flow. Implementations using Message Broker vary significantly from simple routing of messages to complex validation and transformation. Some implementations also read data from a database and use that to populate messages.
- In the complex environments which many of you have it is sometimes difficult to draw functional boundaries. Should an existing application be replaced by logic which is implemented within a message flow or should use of brokers be restricted to connecting and or transforming the messages between communicating applications. This is a difficult issue to resolve. What is clear is that it is the more complex implementations which more often experience performance problems.
- However much function you do decide to implement in a message flow it is important to have clear boundaries between pieces of application processing.
- Where companies have tried to mix logic between brokers and an existing application by retaining some logic in legacy applications poor results have been obtained. That is low throughput and high resource utilization. By trying to mix processing in this way there is the danger of having a series of very expensive function calls. Each call consists of an MQPUT by the message flow, an MQGET by the application, some application processing, an MQPUT by the application and finally an MQGET by another message flow in order to continue processing within the broker. The function calls has resulted in 4 MQPUT and MQGET requests. Each message has also had to be parsed, since context was lost.
- In some situations it will be necessary to retain previous applications. If a message flow must interact with such an application give thought as to how the two will communicate. Keep such communications to a minimum. Maybe processing can be batched, that is multiple calculations can be performed at once.
- It is recommended to establish as soon as possible the processing that MUST be performed in a message flow. Is it mandatory to use persistent messages or can an application cope with retry logic for example. Each of the mandatory requirements typically has an impact on performance and usually for the worse! It often lowers the maximum throughput for a given level of resources or viewed another ways results in the need for more resources to achieve the same level of throughput.
- Whatever the requirements of a particular situation there is usually the potential to have some impact on performance. Putting it another way it is possible to write a flow poorly so resulting in excessive processing costs if you are not careful.
- There are a number of common processing styles with Message Broker. These are shown in the following foils. There are multiple patterns. Often more than one can be combined in a single message flow. These processing patterns perform different functions and will have different processing costs.
- Many of the processing samples which are displayed are now shipped in Message Broker V6. They can be imported and run using sample data that is provided. They can be used as a base for your own message flows. We will now look at some of the most common processing styles.

Common Usage Styles

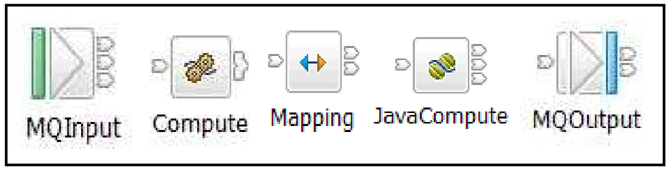
Routing

- Used to redirect messages
- Routing methods:
  - Static rules
  - Dynamic data
    - **Cache**
    - **Database**



Transformation

- Used to perform business processing on a message
- Often combined with other processing patterns



Routing and Transformation

N O T E S

- Routing is used to redirect messages. One or more copies of message can be sent to one or more different destinations. This could involve a change of format or protocol but then the processing starts to become transformation.

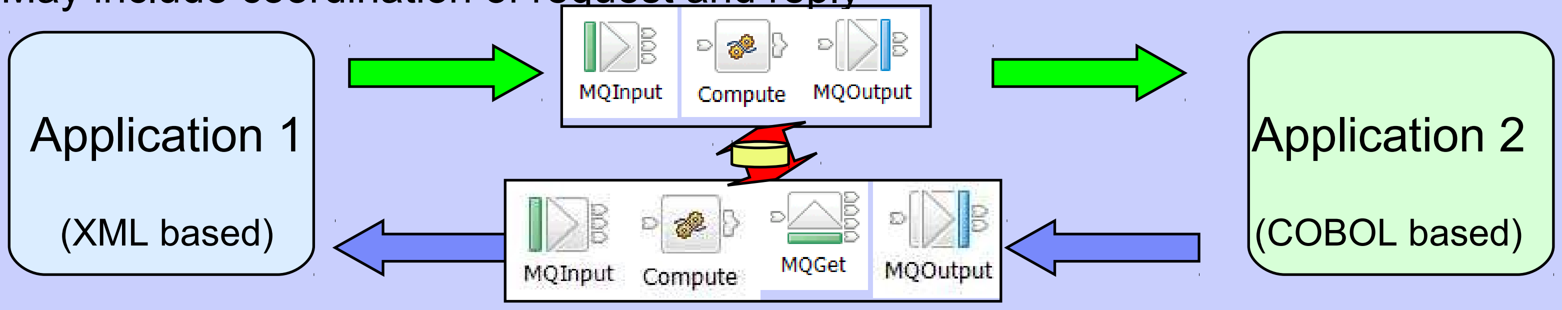
- Routing usually involves looking at a field in the source data header or body. Routing decisions may be statically coded in the message flow. For a more dynamic approach the data values and corresponding locations could be held in an in memory cache or database.

- Transformation processing involves the use of one or more transformation technologies such as Compute, JavaCompute,, XMLT, Mapping node or WTX. In this processing the input message will be processed according to some business rules. There may also be a change of message format or protocol.

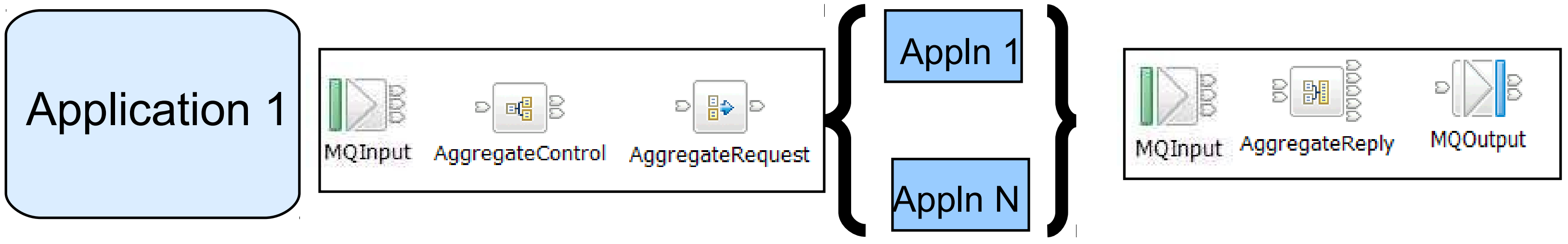
Common Usage Styles

Request Reply

- Commonly used to invoke an existing application
- Often involves XML <> CWF conversion
- May include coordination of request and reply



Aggregation



- Used to invoke one or more back end systems and coordinate replies

Request Reply and Aggregation

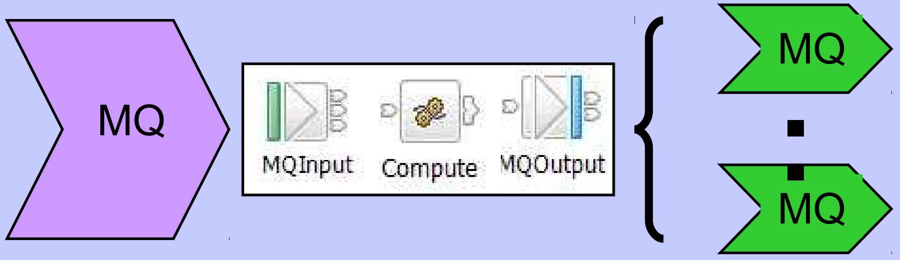
N O T E S

- ❑ Request Reply is probably the most common type of processing. Two applications need to communicate. Here we have Application 1 and Application 2 communicating through two message flows. A common usage case is a new application which is based on the use of XML needs to communicate with an application which uses a COBOL copybook.
- ❑ In this example the first message flow transforms the request message from Application 1 into a format that Application 2 can understand. The output of the first message flow is sent to Application 2 which processes the request and issues a reply. The reply is processed in a second message flow which converts the response message into a format that Application 1 can understand.
- ❑ Some times the request and reply proceed independently. In others every reply must be matched to a request. There are several different ways to do this. You might use:
  - o An MQ queue to store the request message in (done in the request message flow) and then have the reply message flow issue an MQGET to retrieve the original request message. Another flow invoked by the Timer node could periodically monitor for unmatched messages and report them.
  - o A database to store a record of the information in the request message in a similar way to above. The reply message flow might then issue a database read followed by a delete to remove the row holding the request information. Whilst this approach works it is must heavier in processing costs that using the MQ queue based approach. This would result in a lower maximum message rate.
- ❑ Note in some situations there may only be request messages.
- ❑ The Aggregation example is a more complex form of the Request Reply case above. In this case there is additional complexity as all of the replies from the intermediate applications that were invoked must be collected together first before the reply message for the original request can be sent. This is the type of processing that could be used to book a holiday for example. For the holiday we need a flight, hotel, money and a car. We want to know what all have been successful before replying to traveller Aggregation is a good way of achieving this..
- ❑ Whilst you can manage this complexity yourself it is much easier to use the Aggregation nodes provided as part of Message Broker. With Message Broker V6 and later version of Message Broker V5 they are now based on the use of MQ queues. Previously they used the broker database. As a result of the change performance is significantly improved.

Common Usage Styles

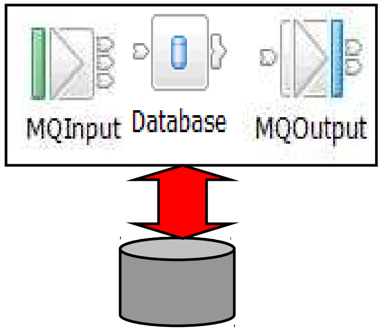
Large Message (or Message Slicing)

- Split one large message into multiple smaller messages



Data Warehousing

- Used to store data in a database



Large Messaging and Data Warehouse

N O T E S

- The Large messaging or message slicing processing involves the splitting of one large message into multiple smaller messages. To achieve this for large messages may require special techniques otherwise virtual memory use can become an issue.

- This type of processing is most easily performed on messages with a repeating structure. The message is usually large because it contains many iterations of a common structure.

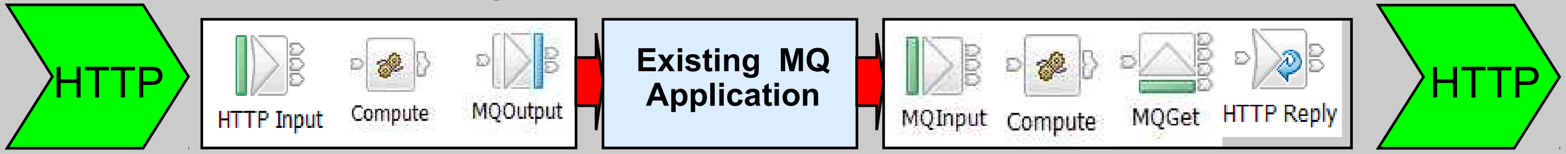
- One technique which works well with a repeating structure is that of using a mutable [changeable] message tree. **Note** InputRoot is not mutable. Each repeating element is processed one at a time. It is parsed into a mutable tree, processed and then deleted. The next element of the structure is then processed. The Large Messaging sample in Message Broker V6 shows how this technique can be used.

- The data warehousing case involves storing data in a database. It could be combined with the Large Messaging case described above, so that a large message is read into the message, split and each repeating element of the structure is inserted into a database.

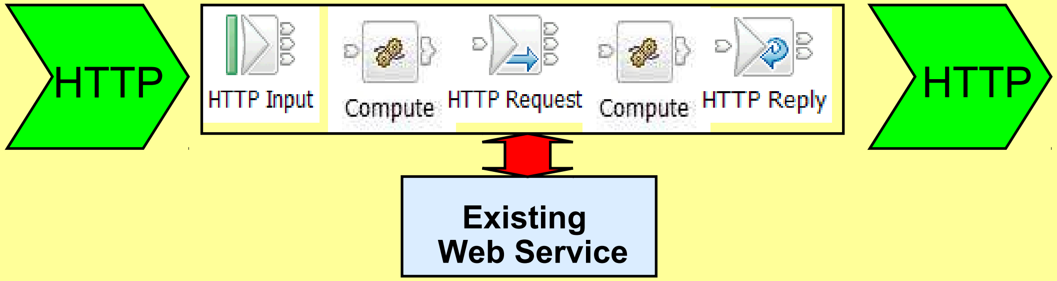
Common Usage Styles

Façading to invoke/expose Web Services

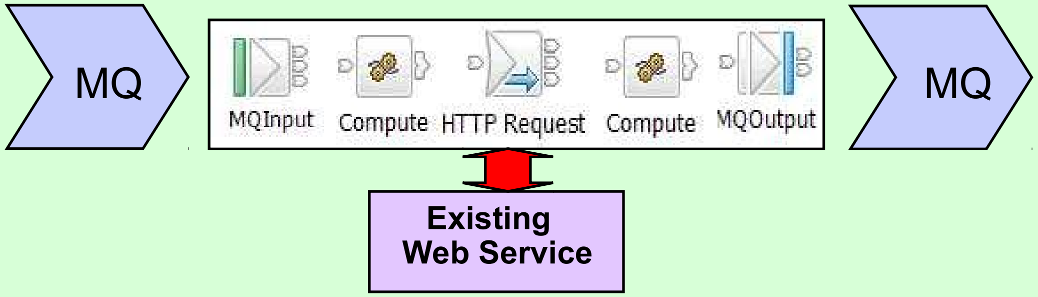
- Façade an existing application



- Façade or invoke an existing web service



- Issue an HTTP Request from within the messaging backbone



Web services

N
O
T
E
S

- The use of web services is becoming increasing popular. The ability to wrapper existing applications and invoke others from within a message flow means that Message Broker has a strong role to play as you web services enable existing applications or as you combine one or more applications and expose them collectively as a single web service.

- In this chart three common uses of Message Broker for web services processing are presented.

  - In the first case a message flow is used to façade an existing application. The existing application which is MQ enabled can be overlapped by two message flows. The first receives the web services request in an HTTPInput node. Some processing is performed on the request and an MQ message written to invoke the existing application. The application performs it processing and sends the reply to the second message flow which formats a reply. The MQGET node is used to retrieve the session identifier of the request which was stored in an MQ message . This session identifier is needed to ensure that the reply is sent to the correct client.

  - In the second case an existing web service is invoked from within a message flow using the HTTPRequest node. This is a synchronous request. This style of processing gives the ability to re-present an existing web service or to add some additional processing for example or to reformat the request prior to invoking the existing web service. In this example all of the processing takes place within a single message flow. As such there is no need to retrieve the session identifier as happened in the case above.

  - The third case shows how an existing web service can be invoked by existing MQ enabled applications. An MQ message is sent to the message flow and received by an MQInput node. The message is transformed into a format suitable to call the web service. The web service is invoked using the HTTPRequest node. This issues a synchronous call. The contents of the reply from the web service are used to build a reply message. This is written to an MQ queue and is read by the next MQ enabled application in the processing sequence.

Contents

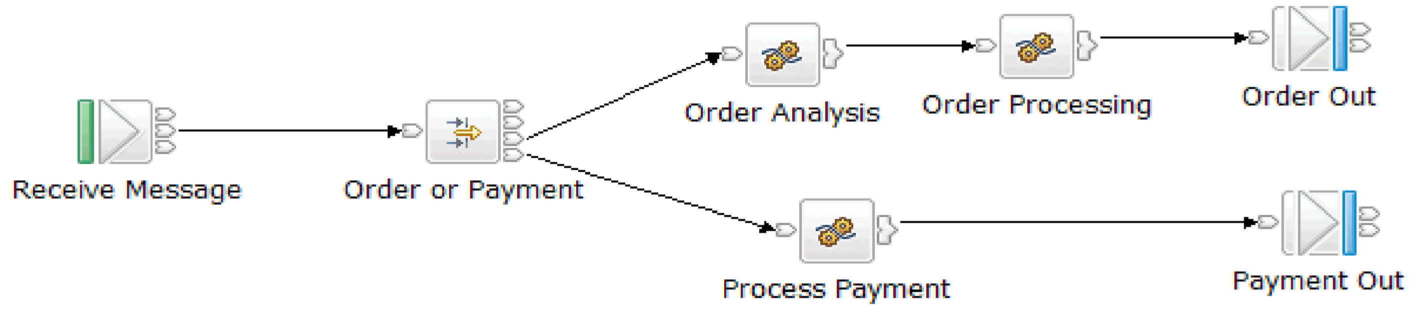
Contents

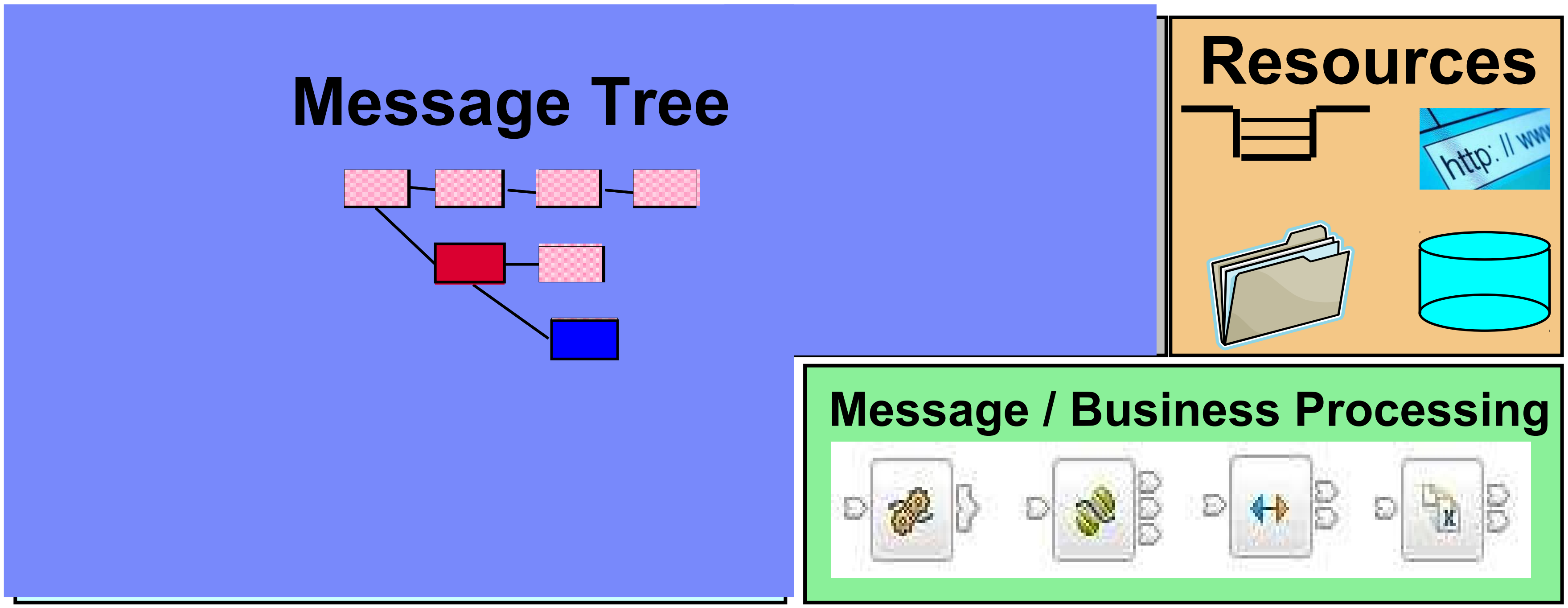
Design Considerations

N
O
T
E
S

- In this sections we will look at some of the key factors which affect message flow processing costs. As well as identifying the issue we will also look at some practical examples of how processing costs can be reduced by adopting different designs.
- But first it is helpful to understand the different areas in which processing costs arise when a message flow is invoked. This is key to the material in this section. So lets take a look under the covers of a simple message flow.

Message Flow Processing Constituent Costs





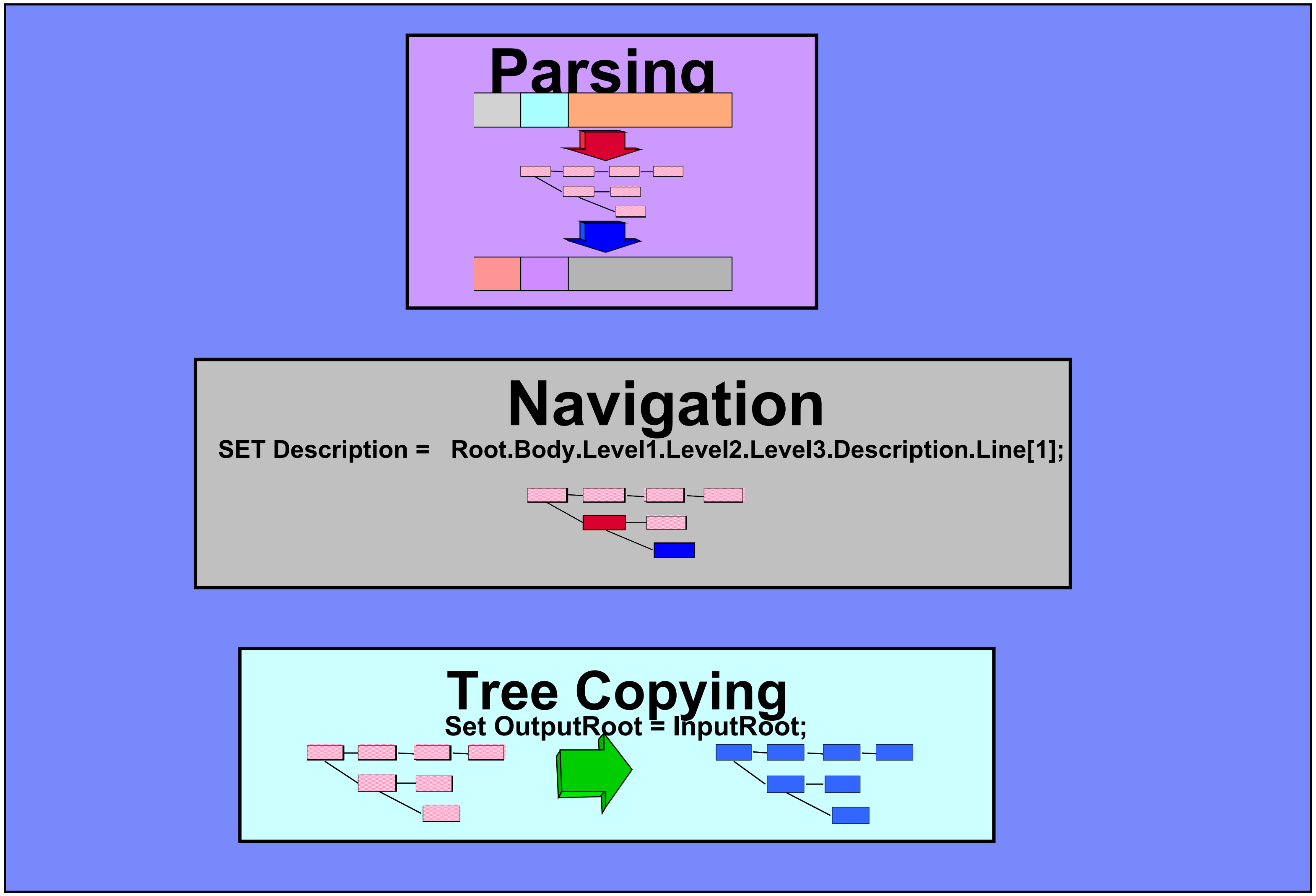
Message Flow Processing Constituent Costs

N O T E S

- This foils shows a simple message flow. When this or any other message flow processes messages costs arise. These costs are:
  - Parsing. This has two parts. The processing of incoming messages and the creation of output messages. Before an incoming message can be processed by the nodes or ESQL it must transformed from the sequence of bytes, which is the input message, into a structured object, which is the message tree. Some parsing will take place immediately such as the parsing of the MQMD, some will take place, on demand, as fields in the message payload are referred to within the message flow. The amount of data which needs to be parsed is dependent on the organization of the message and the requirements of the message flow. Not all message flows may require access to all data in a message. When an output message is created the message tree needs to be converted into an actual message. This is a function of the parser. The process of creating the output message is referred to as serialization or flattening of the message tree. The creation of the output message is a simpler process than reading an incoming message. The whole message will be written at once when an output message is created. We will discuss the costs of parsing in more detail later in the presentation.
  - Message/Business Processing. It is possible to code message manipulation or business processing in any one of a number of transformation technologies. These are ESQL, Java, Mapping node, XSL and WebSphere TX mappings. It is through these technologies that the input message is processed and the tree for the output message is produced. The cost of running this is dependent on the amount and complexity of the transformation processing that is coded.
  - Navigation. This is the process of "walking" the message tree to access the elements which are referred to in the ESQL or Java. The cost of navigation is dependent on the complexity and size of the message tree which is in turn dependent on the size and complexity of the input messages and the complexity of the processing within the message flow.
  - Tree Copying. This occurs in nodes which are able to change the message tree such as Compute nodes. A copy of the message tree is taken for recovery reasons so that if a compute node makes changes and processing in node incurs or generates an exception the message tree can be recovered to a point earlier in the message flow. Without this a failure in the message flow downstream could have implications for a different path in the message flow. The tree copy is a copy of a structured object and so is relatively expensive. It is not a copy of a sequence of bytes. For this reason it is best to minimize the number of such copies, hence the general recommendation to minimize the number of compute nodes in a message flow. Tree copying does not take place in a Filter node for example since ESQL only references the data in the message tree and does not update it.
  - Resources. This is the cost of invoking resource requests such as reading or writing WebSphere MQ messages or making database requests. The extent of these costs is dependent on the number and type of the requests.
- The parsing, navigation and tree copying costs are all associated with the population, manipulation and flattening of the message tree and will be discussed together a little later on.
- Knowing how processing costs are encountered puts you in a better position to make design decisions which minimize those costs. This is what we will cover during the course of the presentation.

Contents

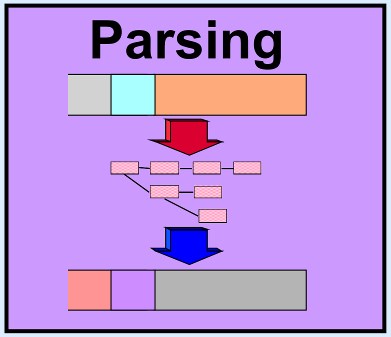
Message Tree Processing



SET Description = Root.Body.Level1.Level2.Level3.Description.Line[1];

Set OutputRoot = InputRoot;

Parsing



- The means of populating and serialising the message tree
  - Multiple parsers available: XMLNSC, MRM XML, CWF, TDS, MIME, JMSMap, JMSStream, BLOB, IDOC, RYO
  - Message complexity varies significantly

...and so do costs!

- Occurs whenever the message body is accessed
- Ways to reduce/eliminate parsing costs
  - Use cheapest parser possible
  - Identify the message type quickly
  - Use Parsing strategies
    - Parsing avoidance
    - Partial parsing
    - Opaque parsing

Design Considerations

N O T E S

- In order to be able to process a message or piece of data within Message Broker we need to be able to model that data and build a representation of it in memory. Once we have that representation we can transform the message to the required shape, format and protocol using transformation processing such as ESQL or Java. That representation of a sequence of bytes into a structured object is the message tree. It is populated though a process called parsing.
- There are two parts to parsing within Message Broker:
  - The first is the most obvious. It is the reading and interpretation of the input message and recognition of tokens within the input stream. Parsing can be applied to message headers and message body. The extent to which it is applied will depend on the situation. As parsing is an expensive process in CPU terms it is not something that we want to automatically apply to every part of every message that is read by a message flow. In some situations it is not necessary to parse all of an incoming message in order to complete the processing within the message flow. A routing message flow may only need to read a header property such as user identifier or ApplIdentifyData in the MQMD for example. If the input message is very large parsing the whole message could result in a large amount of additional CPU being unnecessarily consumed if that data is not required in the logic of the message flow.
  - The second, is the flattening, or serialisation of a message tree to produce a set of bytes which correspond to the wire protocol of a message in the required output format and protocol.
- Message Broker provides a range of parsers. The parsers are named on the foil. The parsers cover different message formats. There is some overlap – XMLNSC and MRM XML are both able to parse a generic XML message but there are also differences. MRM XML can provide validation. XMLNSC does not. Other parsers such as JMSStream are targeted at specific message formats. In this case it is a JMS stream message.
- Messages vary significantly in their format and the level of complexity which they are able to model. Tagged Delimited string (TDS) messages for example can support nested groups. More complex data structures make parsing costs higher. The parsing costs of different wire formats is different. You are recommended to refer to the performance reports in order to get information on the parsing costs for different types of data.
- Whatever the message format there are a number of techniques that can be employed to reduce the cost of parsing. We will now look at the most common ones. First though we will discuss the reasons for the different cost of parsing.

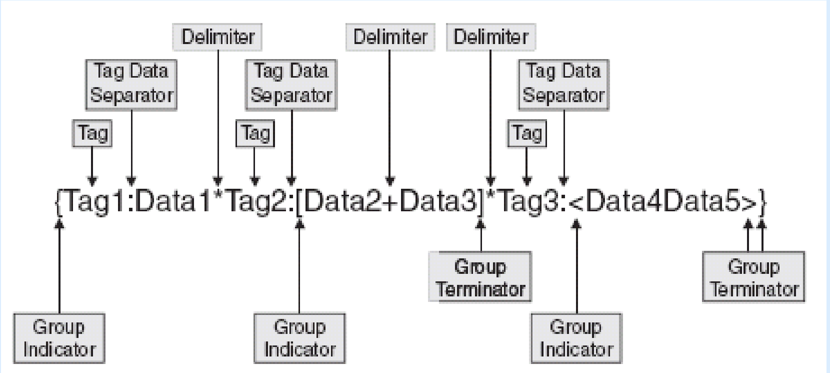
Cost of Parsing

- XML
  - Self describing with simple syntax
  - Must have a closing tag
  - Elements must be properly nested
  - Must have one root element
  - Elements have relationships

```
<Parent>
      <Item>
      <Description>Twister</Description>
      <Category>Games</Category>
      <Price>00.30</Price>
      <Quantity>01</Quantity>
      </Item>
</Parent>
```

- Message formats vary
  - Self describing/modelled
  - Syntax, delimiters
  - Constructs supported
  - So do costs

- TDS
  - Not self describing
  - High degree of flexibility to define messages
  - Supported features
    - **Tags**
    - **Group indicators and terminators**
    - **Fixed length strings**
    - **Fixed length tags**
    - **Separation types**
      - **Fixed length, tagged, delimited, data pattern**
    - **Regular expressions**



Cost of Parsing

N O T E S

- The ease (and thereby processing cost) with which a message can be parsed depends on a number of factors such as:
  - How difficult it is to detect data elements
  - The complexity of structures that are supported
  - Need to validate relationships between parts of the message.

  Given the differences in format it is not surprising that there are also differences in the cost of parsing different message formats.

- As an illustration we have two different formats represented on the foil. XML and Tagged Delimited Strings(TDS). Both are character only formats. We can see that there are a number of significant differences between the two formats. XML is self describing. TDS is not self describing and needs to defined in a message model.

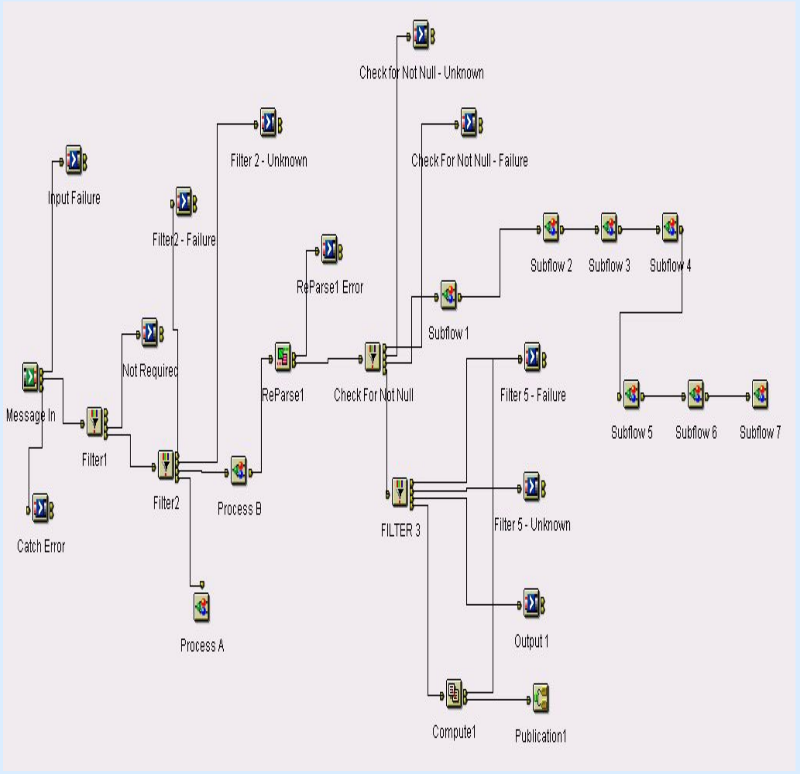
- With XML it is easy to identify when an element starts and when it finishes. The '<' and '>' characters are the only supported tag delimiter characters. Each element must have an opening and closing tag. There must be a single root element. It is easy to see what the message structure is and what the relationship between elements is for example. With such rules parsing is a relatively straightforward issue.

- TDS is quite different in nature. For a start the message is not self describing. There is a high degree of flexibility in the way in which you can define messages. For that we can also read increased processing complexity to successfully parse the message. It is not wrong. This is a trade-off. You can model complex structures but inevitably there is a price to pay. TDS also supports multiple types of separators which allow data element groups to be modelled. Additionally groups can be embedded. Data patterns and regular expressions are also supported. This is clearly more advanced than the XML support. As a consequence the cost of parsing a TDS message is higher.

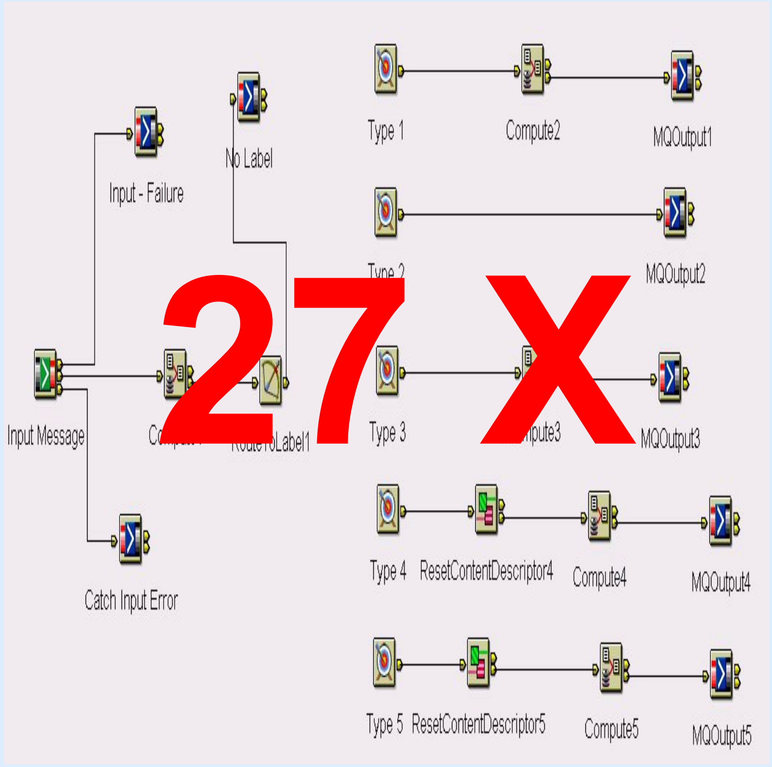
- We have looked at two formats. Other exist each with their own strengths and bias towards modelling a particular type of message. In some situations there is no alternative but to use a particular type of parser. In others there may be a choice. Where there is a choice choose the cheapest.

Identifying the Message Type Quickly

- **Avoid multiple parses to find the message type**



VS



5 msgs/sec

138 msgs/sec

Identifying the Message Type Quickly

N O T E S

- It is important to be able to correctly recognise the correct message format and type as quickly as possible. In message flows which process multiple types of message this can be a problem. When often happens is that the message needs to be parsed multiple times in order to ensure that you have the correct format. Dependent on the particular format and the amount of parsing needed this can be expensive.

- This example shows how a message flow which processed multiple types of input message was restructured to provide a substantial increase in message throughput.

- Originally the flow was implemented as a flow with a long critical path. Some of the most popular messages were not processed until late in the message flow resulting in a high overhead. The flow was complex. This was largely due to the variable nature of the incoming messages, which did not have a clearly defined format. Messages had to be parsed multiple times.

- The message flow was restructured. This substantially reduced the cost of processing. The two key features of this implementation were to use a very simple message set initially to parse the messages and the use of the RouteToLabel and Label nodes within the message flow in order to establish processing paths which were specialized for particular types of message.

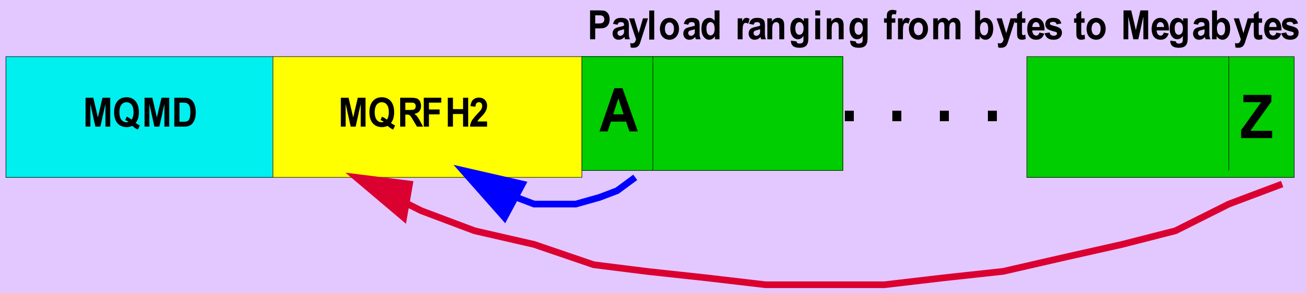
- Because of the high processing cost of the initial flow it was only possible to process 5 non persistent message per second using one copy of the message flow. With the one copy of the restructured flow it was possible to process 138 persistent messages per second on the same system. This is a 27 times increase in message throughput plus persistent messages where now being processed whereas previously they were non persistent.

- By running five copies of the message flow it was possible to:
  - **Process around 800 non persistent messages per second when running with non persistent messages.**
  - **Process around 550 persistent messages per second when running with persistent messages.**

Parsing Avoidance

- Removing the need to parse a body of data

- Promote/copy data to MQMD, MQRFH2 or JMS Properties
  - May save having to parse the message body
  - Particularly useful for message routing



- Consider sending only changed data
  - If it isn’t there it can't be parsed!
  - More application logic but can reduce volume of data to transport and parse

Parsing Avoidance

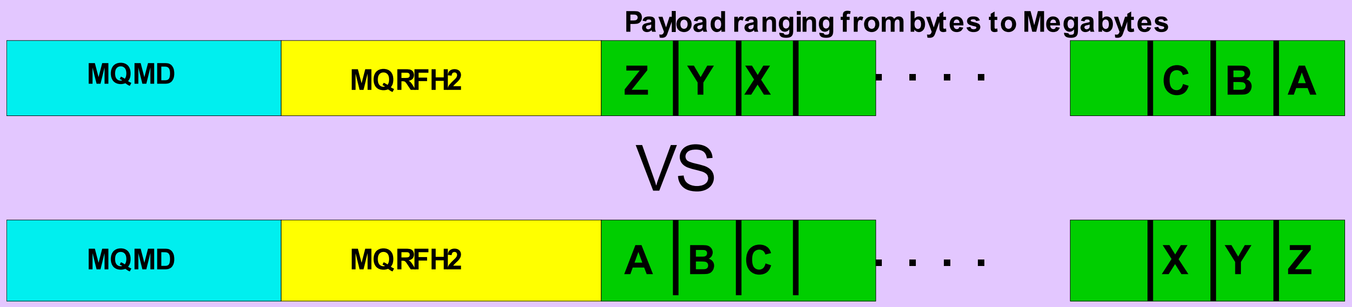
N O T E S

- One very effective technique to reduce the cost of parsing is not to do it. A couple of ways of doing this are discussed in this foil.
- The first approach is to avoid having to parse some parts of the message. Suppose we have a message routing message flow which needs to look at a field in order to make a routing decision. If that field is in the body of the message the body of the incoming message will have to be parsed to get access to it. The processing cost will vary dependent on which field is needed. If it is field A that is OK as it is at the beginning of the body and would be found quickly. If it is field Z then the cost could be quite different, especially if the message is several megabytes in size. A technique to reduce this cost would be to have the application which creates this message copy the field that is needed for routing into a header within the message, say in an MQRFH2 header for an MQ message or as a JMS property if it is a JMS message. If you were to do this it would no longer be necessary to parse the message body so potentially saving a large amount of processing effort. The MQRFH2 heard or JMS Properties folder would still need to be parsed but this is going to be smaller amount of data. The parsers in this case are also more efficient then the general parser for a message body as the structure of the header is known.
- A second approach to not parsing data is to not send it in the first place. Where two applications communicate consider sending only the changed data rather than sending the full message. This requires additional complexity in the receiving application but could potentially save significant parsing processing dependent on the situation. This technique also has the benefit of reducing the amount of data to be transmitted across the network.

Partial Parsing

- Parsing only as far as is needed
  - Explicit(ESQL) VS Implicit(XPath)

- Careful ordering of message fields can reduce the cost



CPU cost ratio	1K msg	16K msg	256K msg
Filter first	1	1	1
Filter last	1.4	3.4	5.6

Measurements taken on pSeries 570 Power 5 with 8 * 1.5 GHZ processors. WebSphere Message Broker V6

Partial Parsing

N O T E S

- Given parsing of the body is needed the next technique is to only parse what is needed rather than parse the whole message just in case. The parsers provided with Message Broker all support partial parsing.
- The amount of parsing which needs to be performed will depend on which fields in a message need to accessed and the position of those fields in the message. We have two example here. One with the fields ordered Z to A and the other with them ordered A to Z. Dependent on which field is needed one of the cases will be more efficient than the other. Say we need to access field Z then the first case is best. Where you have influence over message design ensure that information needed for routing for example is placed at the start of the message and not at the end of the message.
- As an illustration of the difference in processing costs consider the table in the foil. The table shows a comparison of processing costs for two versions of a routing message flow when processing several different message sizes.
- For the filter first test the message flow routes the message based on the first field in the message. For the filter last the routing decision is made based on the last field of the message.
- The CPU costs of running with each message size have been normalised so that the filter first test for each message represents a cost of 1. The cost of the filter last case is then given as a ratio to the filter first cost for that same message size.
- For the 1K message we can see that by using the last field in the message to route the CPU cost of the whole message flow was 1.4 times the cost of using the first field in the same message. This represents the additional parsing which needs to be performed to access the last field.
- For the 16K message, the cost of using the last field in the message had grown to 3.4 times that of the first field in the same message. That is we can only run at one third of the message that we can when looking at the first field of the message.
- For the 256K message, the cost of using the last field in the message was 5.6 times that of the first field in the same message. That is we can only run at one fifth of the message that we can when looking at the first field of the message.
- You can see how processing costs quickly start to grow and this was a simple test case. With larger messages the effect would be even more pronounced. Where the whole message is used in processing this is not an issue as all of the data will need to be parsed at some point. For simple routing cases though the ordering of fields can make a significant difference.
- When using ESQL, and Mapping nodes the field references are typically explicit. That is we have references such as InputRoot.Body.A. Message Broker will only parse as far as the required message field to satisfy that reference. It will stop at the first instance. When using XPath, which is a query language the situation is different. By default an XPath expression will search for all instances of an element in the message which implicitly means a full parse of the message. If you know there is only one element in a message there is the chance to optimise the XPath query to say only retrieve the first instance. See the ESQL and Java coding tips later in the presentation for more information.

Opaque Parsing

- Model the whole of an XML sub tree as a single element in the message tree containing that part of the bit stream
- Reduces message tree size and parsing costs
- Cannot reference the sub tree in message flow processing

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Header/>
 <soapenv:Body>
     <p56:requestAssessorAvailability xmlns:p56="http://ws3.st.mqsi.ibm.com/Hub/Docliteral1">
                <p56:claimID>77021964</p56:claimID>
                <p56:carDetails>
                           <p56:makeOfCar>porshe</p56:makeOfCar>
                           <p56:registration>HK03SSS</p56:registration>
                </p56:carDetails>
                <p56:assArray>
                          <p56:assessors>...</p56:assessors>
                                ...     ...
                                ...     ...
                          <p56:assessors>...</p56:assessors>
                </p56:assArray>
     </p56:requestAssessorAvailability>
     <p56:driverID>2739694321</p56:driverID>
 </soapenv:Body>
</soapenv:Envelope>
```

Opaque Parsing

N O T E S

- Opaque parsing is a technique that allows the whole of an XML sub tree to placed in the message tree as a single element. The entry in the message tree is the bit stream of the original input message. This technique has two benefits:
  - It reduces the size of the message tree since the XML sub tree is not expanded into the individual elements.
  - The cost of parsing is reduced since less of the input message is expanded as individual elements and added to the message tree.
- Use opaque parsing where you do not need to access the elements of the sub tree, for example you need to copy a portion of the input tree to the output message but may not care about the contents in this particular message flow. You accept the content in the sub folder and have no need to validate or process it in any way.
- Opaque parsing is supported for the XMLNS and XMLNSC domains only.
  - The support for the XMLNS domain was introduced in Message Broker V6 fixpack 1 and was an early version of opaque parsing and as such could be changed in the future.
  - The support for the XMLNSC domain was added in Message Broker V6.1. It has a different interface from the XMLNS domain to specify the element names which are to be opaquely parsed.
- The support for the XMLNS domain continues to be available in its original form in Message Broker V6.1. Indeed this is the only way in which the messages in the XMLNS domain can be opaquely parsed. It is not possible to use the same interface that is provided for the XMLNSC to specify element names for the XMLNS domain.
- The following pages of notes explain how opaque parsing is performed for messages in the XMLNS and XMLNSC domains.

Opaque Parsing for the XMLNS domain

N O T E S

- Opaque parsing is achieved by an overload of the CREATE statement. The FORMAT qualifier of the PARSE clause is set to the case-sensitive string XMLNS_OPAQUE and the TYPE qualifier of the PARSE clause is set to the name of the XML element which is to be parsed in an opaque manner. **Note**: It is only available from within ESQL and is supported for the XMLNS domain only.
- In the example below we want to route a message based on the element `<p56:driverID>` (see previous foil for message format). This is marked in red on the foil. By default if ESQL referred to this field all of the message prior to it would be parsed and instantiated in the message tree. Suppose the element `<p56:requestAssessorAvailability>` was large with many child elements with it. In this case the cost of populating the message tree would be large. As no part of `<p56:requestAssessorAvailability>` is needed in the message flow we can opaquely parse this element. The ESQL to do this is shown below.

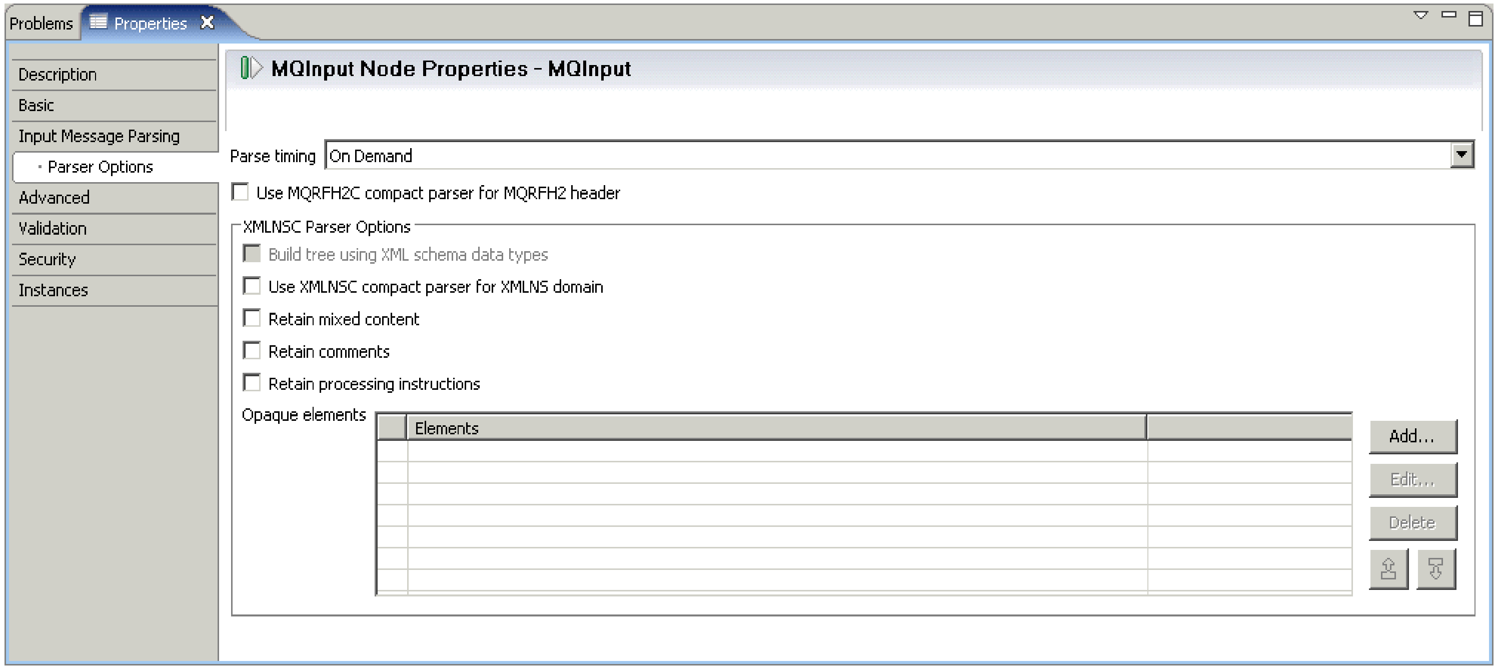
```
DELCARE p56  NAMESPACE ‘http://ws3.st.mqsi.ibm.com/Hub/Docliteral1’;

DECLARE BitStream BLOB ASBITSTREAM(InputRoot.XMLNS
                                   ENCODING InputRoot.Properties.Encoding
                                   CCSID InputRoot.Properties.CodedCharSetId);
--Namespace Prefix
CREATE LASTCHILD OF OutputRoot
 DOMAIN(’XMLNS’)
             PARSE (BitStream
                  ENCODING InputRoot.Properties.Encoding
              CCSID InputRoot.Properties.CodedCharSetId
              FORMAT ’XMLNS_OPAQUE’
                  TYPE ’p56:requestAssessorAvailability ’);

--Namespace URI
CREATE LASTCHILD OF OutputRoot
 DOMAIN(’XMLNS’)
             PARSE (BitStream
                  ENCODING InputRoot.Properties.Encoding
              CCSID InputRoot.Properties.CodedCharSetId
              FORMAT ’XMLNS_OPAQUE’
                 TYPE ’{http://ws3.st.mqsi.ibm.com/Hub/Docliteral1}requestAssessorAvailability’);
```

- For more information on opaque parsing consult the V6 Fixpack 1 version of the ESQL manual on page 126 which is available at ftp://ftp.software.ibm.com/software/integration/wbibrokers/docs/V6.0/messagebroker_ESQL.pdf.

Opaque Parsing for the XMLNSC domain

N O T E S

- The message domain must be set to XMLNSC.
- The elements in the message which are to be opaquely parsed are specified on the 'Parser Options' page of the input node of the message flow. See the picture below of the MQInput node in Message Broker V6.1.
- Enter those element names that you wish to opaquely parse in the 'Opaque Elements' table.
- Be sure not to enable message validation as it will automatically disable opaque parsing. Opaque parsing in this case does not make sense since for validation the whole message must be parsed and validated.
- Opaque parsing for the named elements will occur automatically when the message is parsed.
- It is not currently possible to use the CREATE statement to opaquely parse a message in the XMLNSC domain.



MQInput Node Properties - MQInput

Parse timing: On Demand

☐ Use MQRFH2C compact parser for MQRFH2 header

XMLNSC Parser Options
☐ Build tree using XML schema data types
☐ Use XMLNSC compact parser for XMLNS domain
☐ Retain mixed content
☐ Retain comments
☐ Retain processing instructions

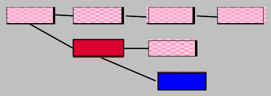
Opaque elements

Elements

Navigation

Navigation

```
SET Description =   Root.Body.Level1.Level2.Level3.Description.Line[1];
```



- The means of walking the message tree
  - Occurs in Compute, JavaCompute, PHPCompute and Mapping Nodes

- Ways to reduce navigation costs
  - Build smaller Message Tree
    - Smaller messages,
    - Use Compact Parsers (XMLNSC, MRM XML, RFH2C)
    - Opaque parsing, partial parsing
  - ESQL/Java
    - Use reference variables/pointers

Navigation

N O T E S

- Navigation is the process of accessing elements in the message tree. It happens when you refer to elements in the message tree in the Compute, JavaCompute and Mapping nodes.

- The cost of accessing elements is not always apparent and is difficult to separate from other processing costs.

- The path to elements is not cached from one statement to another. If you have the ESQL statement

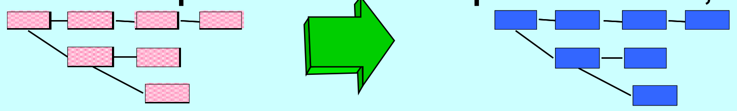
  SET Description = Root.Body.Level1.Level2.Level3.Description.Line[1]; the broker runtime will access the message tree starting at the correlation Root and then move down the tree to Level1, then Level2, then Level3, then Description and finally it will find the first instance of the Line array. If you have this same statement in the next line of your ESQL module exactly the same navigation will take place. There is no cache to the last referenced element in the message tree. This is because access to the tree is intended to be dynamic to take account of the fact that it might change structure from one statement to another. There are techniques available though to reduce the cost of navigation.

- With large message trees the cost of navigation can become significant. There are techniques to reduce which you are recommended to follow:
  - Use the compact parsers (XMLNSC, MRM XML and RFH2C). The compact parsers discard comments and white space in the input message. Dependent on the contents of your messages this may have an effect of not. By comparison the other parsers include data in the original message, so white space and comments would be inserted into the message tree.
  - Use reference variables if using ESQL or reference pointers if using Java. This technique allows you to save a pointer to a specific place in the message tree. Say to Root.Body.Level1.Level2.Level3.Description for example.
- For an example of how to use reference variables and reference pointers see the coding tips given in the Common Problems section.

Message Tree Copying

Tree Copying

Set OutputRoot.XML.A = InputRoot.XML.A;



- The process of copying the message tree (whole or part)
  - Occurs in Compute, JavaCompute, PHPCompute, Mapping Nodes

- Ways to reduce tree copying costs
  - Produce a smaller message tree
    - Smaller messages,
    - Use Compact Parsers (XMLNSC, MRM XML, RFH2C)
    - Opaque parsing, partial parsing
  - Reduce the number of times the tree is copied
    - Reduce the number of Compute and JavaCompute nodes in the flow
      - See if Compute mode not using `message` is possible (see notes)
    - Copy at the highest level in the tree that is possible
    - Copy data to the Environment correlation (remember it is not backed out)

Tree Copying

N O T E S

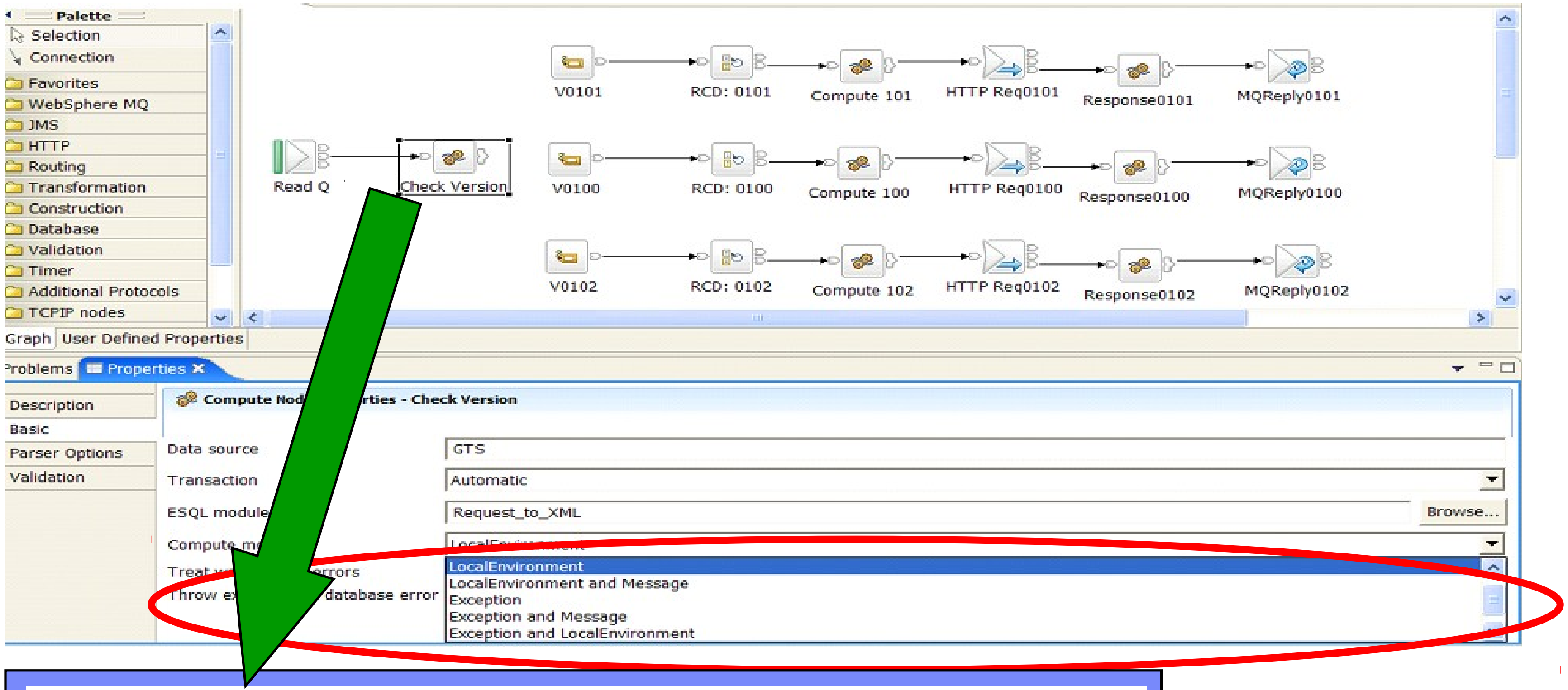
- Tree Copying is the process of copying the message tree either in whole or part.

- Copying occurs in nodes which are able to change the message tree such as Compute nodes. A copy of the message tree is taken for recovery reasons so that if a compute node makes changes and processing in node incurs or generates an exception the message tree can be recovered to a point earlier in the message flow. Without this a failure in the message flow downstream could have implications for a different path in the message flow. The tree copy is a copy of a structured object and so is relatively expensive. It is not a copy of a sequence of bytes. For this reason it is best to minimize the number of such copies, hence the general recommendation to minimize the number of compute nodes in a message flow. Tree copying does not take place in a Filter node for example since ESQL only references the data in the message tree and does not update it.

- There are a few ways to reduce the costs of tree copying. These are:
  - Produce a smaller message tree in the first place. A smaller tree will cost less to copy. Ways to achieve this are to use Smaller messages, use Compact Parsers (XMLNSC, MRM XML, RFH2C), use Opaque parsing partial parsing (all discussed previously).
  - Reduce the number of times the whole tree is copied. Reducing the number of compute nodes (ESQL or Java) will help to reduce the number of times that the whole tree needs to be copied. In particular avoid situations where you have one compute node followed immediately by another. In many cases you may need multiple computes across a message flow. What we want to do is to optimise the processing not have to force everything into one compute node.
  - Copy portions of the tree at the branch level if possible rather than copying individual leaf nodes. This will only work where the structure of the source and destination are the same but it is worth doing if possible.
  - Copy data to the Environment correlation (remember it is not backed out) and work with it in Environment. This way the message tree does not have to be copied every time you run a compute node. Environment is a scratchpad area which exists for each invocation of a message flow. The contents of Environment are accessible across the whole of the message flow. Environment is cleared down at the end of each message flow invocation.
    - The first compute node in a message flow can copy InputRoot to Environment. Intermediate nodes then read and update values in the Environment instead of using the InputRoot and OutputRoot correlations as previously.
    - In the final compute node of the message flow OutputRoot must be populated with data from Environment. The MQOutput node will then serialize the message as usual. Serialization will not take place from Environment.

Whilst use of the Environment correlation is good from a performance point of view be aware that the any updates made by a node which subsequently generates an exception will remain in place. There is no back-out of changes as there is when a message tree copy was made prior to the node exception

Example of Avoiding a Tree Copy



Compute Node Properties - Check Version

Data source: GTS

Transaction: Automatic

ESQL module: Request_to_XML

Compute mode: LocalEnvironment

Treat warnings as errors

Throw exception on database error

LocalEnvironment
LocalEnvironment and Message
Exception
Exception and Message
Exception and LocalEnvironment

```
-- Set the version numbers this flow supports
SET Environment.SupportedVersions = '0102 0101 0100';

-- getMessageVersion returns label name for the message version (for example, '0102')
PROPAGATE TO LABEL 'V' || getMessageVersion(Environment, InputRoot, TRUE);

-- PROPAGATE already passed the message; do not propagate to "Out"
RETURN FALSE;
```

Example of Avoiding a Tree Copy

N O T E S

- This foil shows one case where it was possible to avoid a tree copy. You may be able to use this same technique in other situations.

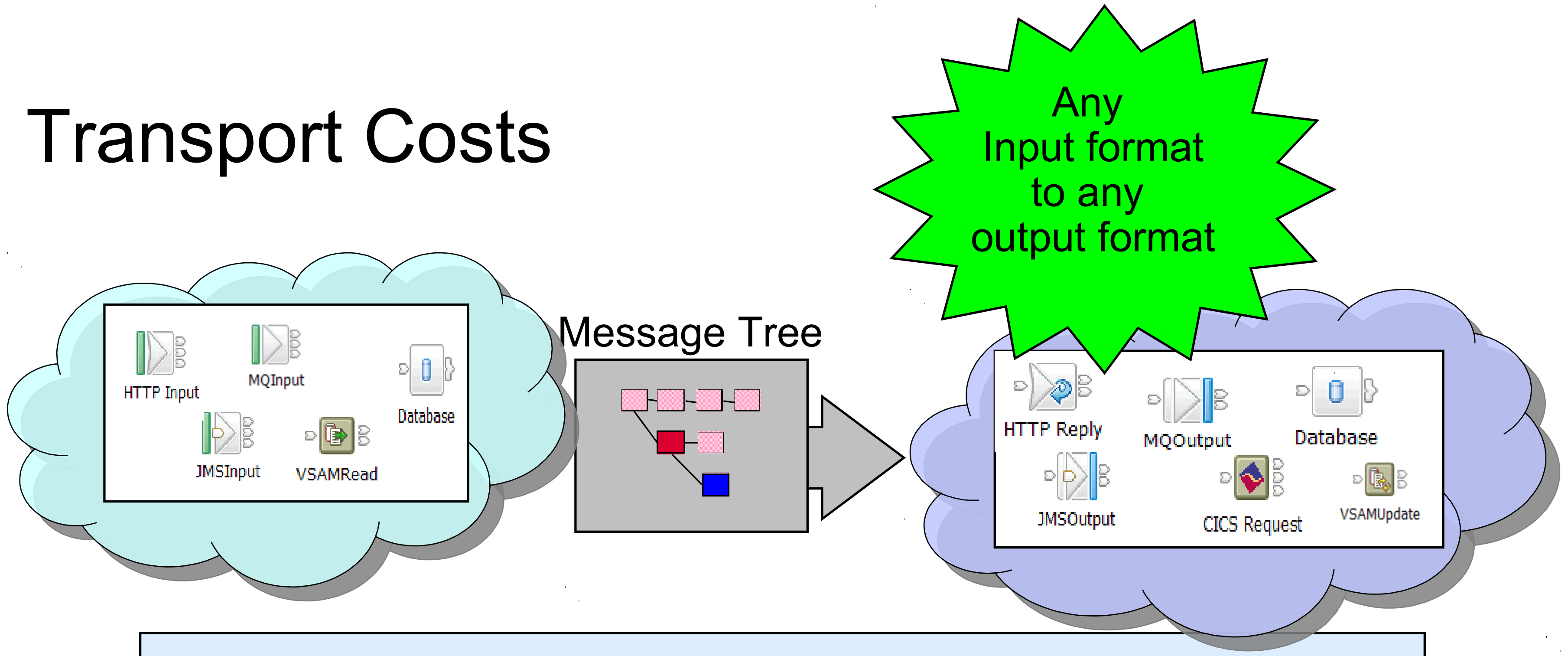
- The purpose of the message flow is read MQ messages, determine the message type and route to the appropriate type of processing. There are three legs of processing, each starting with a Label node (V0100, V0101 & V0102).

- The compute node Check_Version issues a PROPAGATE to the appropriate label node dependent on a field in the incoming message. The message version information is obtained from the incoming message by a function called getMessageVersion. The ESQL on the foil shows how this is done.

- As processing at this point is limited to routing only there is no need for Check_Version to modify the incoming message tree and as such a value of LocalEnvironment is sufficient for the Compute Mode of the node. Had a value of LocalEnvironment and Message been selected then we would have needed to copy the message tree which would have increased processing costs, unnecessarily in this case.

Contents

Transport Costs



- Reliability of the transport
  - Does the message flow need to make 'good' – e.g. Using an MQ persistent message to persist data from TCP/IP client
- Performance
  - Different protocols have different performance characteristics
  - Session initiation and management costs
    - MQCONN per request, HTTP 1.1 persistent sessions etc
- Separation of transport and data

Transport Costs

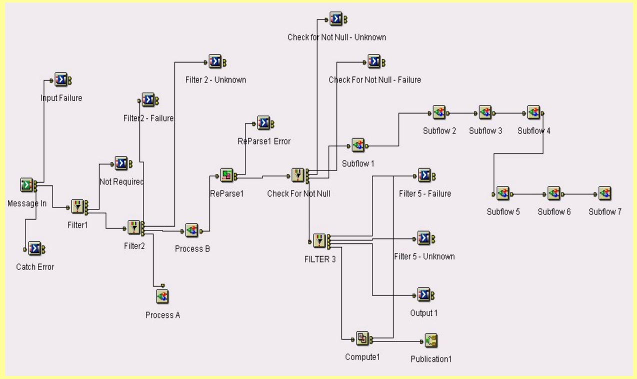
N O T E S

- There are a wide variety of ways of reading data into Message Broker and sending it out once it is has been processed. The most common methods are MQ messages, JMS messages, HTTP and files. Collectively for this discussion lets refer to them as methods of transport although we recognise that is pushing it where file, VSAM and database are concerned.
- Different transports have different reliability characteristics. An MQ persistent message has assured delivery. It will be delivered once and once only. MQ non persistent messages are also pretty reliable but are not assured. If there is a failure of machine or power outage for example data will be lost with these non persistent messages. Similarly data received over a raw TCP/IP session will be unreliable. With WebSphere MQ Real-time messages can be lost if there is a buffer overrun on the client or server for example.
- In some situations we are happy to accept the limitations of the transport in return for its wide availability or low cost for example. In others we may need to make good the shortcoming of the transport. One example of this is the use of a WebSphere MQ persistent message (or database insert) to reliably capture coming over a TCP/IP connection as the TCP/IP session is unreliable – hey this is why MQ is popular because of the value it builds on an unreliable protocol.
- As soon as data is captured in this way the speed of processing is changed. We now start to work at the speed of MQ persistent messages or the rate at which data can be inserted into a database. This can result in a significant drop in processing dependent on how well the queue manager or database are tuned.
- In some cases it is possible to reduce the impact of reliably capturing data if you are prepared to accept some additional risk. If it is acceptable to have a small window of potential loss you can save the data to an MQ persistent message or database asynchronously. This will reduce the impact of such processing on the critical path. This is a decision only you can make. It may well vary by case by case dependent on the value of the data being processed.
- It is important to think how data will be received into Message Broker. Will data be received one message at a time or will it arrive in a batch. What will be involved in session connection set-up and tear-down costs for each message or record that is received by the message flow. Thinking of it in MQ terms will there be an MQCONN, MQOPEN, MQPUT, MQCMT and MQDISC by the application for each message sent to the message flow. Lets hope not as this will generate lot of additional processing for the broker queue manager [assuming the application is local to the broker queue manager]. In the HTTP world you want to ensure that persistent sessions (not to be confused with MQ persistence) are used for example so that session set-up and tear-down costs are minimised. Make sure that you understand the sessions management behaviour of the transport that is being used and that you know how to tune the transport for maximum efficiency.
- Message Broker give you the ability to easily switch transport protocols, so data could come in over MQ and leave the message flow in a file or be sent as a JMS message to any JMS 1.1 provider. In your processing you should aim to keep a separation between the transport used and the data being sent over that transport. This will make it easier to use different transports in the future.

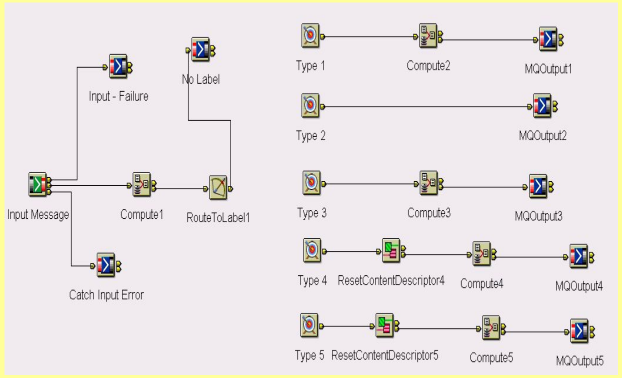
Contents

Message Flow Structure

- Specific VS Generic



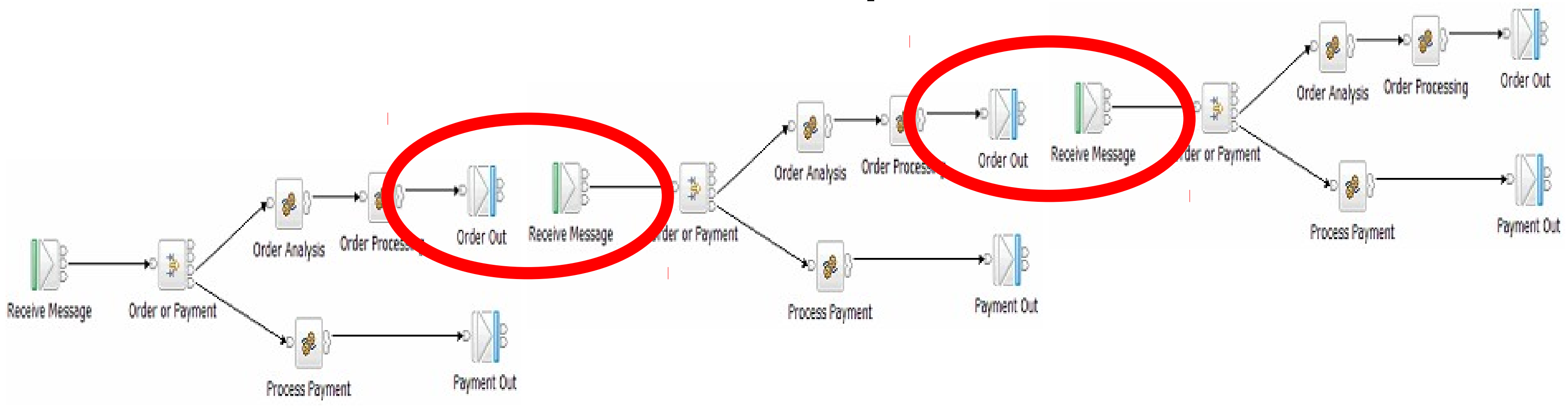
- Clarity
  - RouteToLabel and Label Nodes



- Reuse
  - Subflows
  - ESQL Procedures/Functions
  - Java Classes
  - Stored Procedure

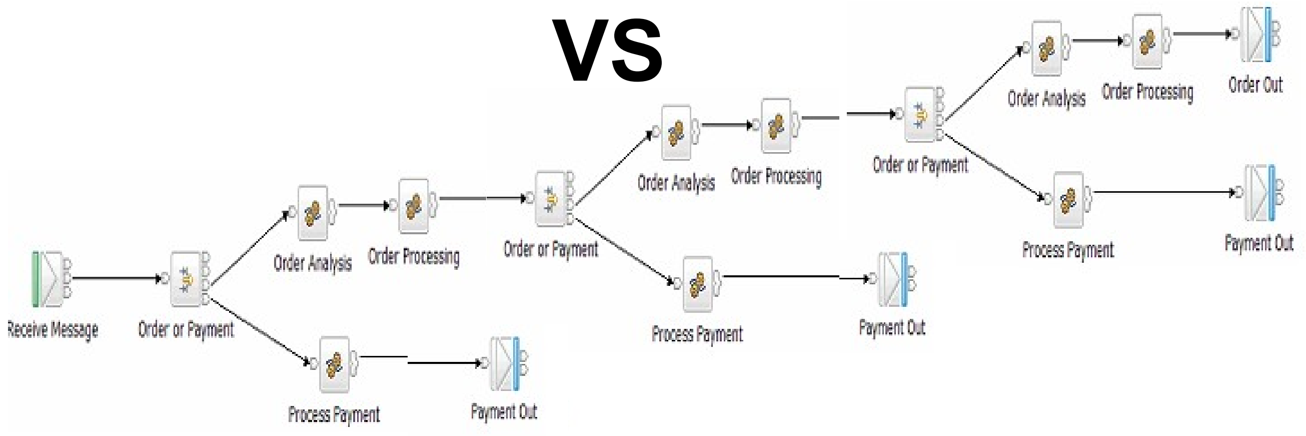
- One VS multiple flows

Number of Flows in Sequence



3X Read Msg – 3X Parse – 3XProcess – 3XSerialise – 3XWrite Msg

VS



Read Msg - Parse – 3X(Process) - Serialise – Write Msg

Save: 2X Read Msg, 2X Parse, 2X Serialise, 2X Write Msg

Message Flow Structure

N O T E S

- It is important to think about the structure of your message flows and to think about how they will process incoming data. Will one message flow process multiple message types or will it process a single type only. Where a unique message flow is produced for each different type of message it is referred to as a specific flow. An alternative approach is to create message flows which are capable of processing multiple message types. There may be several such flows, each processing a different group of messages. We call this a generic flow.
- There are advantages and disadvantages for both specific and generic flows.
  - With specific flows there are likely to be many different message flows. In fact as many message flows as there are types of input message. This does increase the management overhead which is a disadvantage. A benefit of this approach though is that processing can be optimized for the message type. The message flow does not have to spend time trying to determine the type of message before then processing it.
  - With the generic flow there is a processing cost over and above the specific processing required for that type of message. This is the processing required to determine which of the possible message types a particular message is. Generic flows can typically end up parsing an input message multiple times. Initially to determine the type and then subsequently to perform the specific processing for that type of message. A benefit of the generic flow is that there are likely to be fewer of them as each is able to process multiple message types. This makes management easier. Generic flows tend to be more complex in nature. It is relatively easy to add processing for another type if message though. With generic message flows it is important to ensure that the most common message types are processed first. If the distribution of message types being processed changes over time this could mean processing progressively becomes less efficient. You need to take account of this possibility in you message flow design. Use of the RouteToLabel node can help with this problem.
- There are advantages to both the specific and generic approaches. From a message throughput point of view it is better to implement specific flows. From a management and operation point of view it is better to use generic flows. Which approach you choose will depend on what is important in your own situation.

- To bring clarity to your message flow processing consider using the RouteToLabel node. It can help add structure and be effective in separating different processing paths with a message flow. Without flows can become long and unwieldly resulting in high processing costs for those messages with have a long processing path. See the two examples on the foil. Both perform the same processing but have different structures. One is clearly more readable than the other.

- There are several ways of achieving code reuse with Message Broker. They are subflows, ESQL procedures and functions, invoking external Java classes and calling a database stored procedure. There are some performance implications of using the different techniques which we will briefly look at.

Message Flow Structure

N O T E S

- Subflow's are a facility to encourage the reuse of code. Subflow's can be embedded into message flows. The subflow's are 'in-lined' into the message flow at deploy time. There are no additional nodes inserted into the message flow as a result of using subflow's. However be aware of implicitly adding extra nodes into a message flow as a result of using subflow's. in some situations compute nodes are added to subflow's to perform marshalling of data from one part of the message tree into a known place in the message tree so that the data can be processed by the message flow. The result may then be copied to another part of the message tree before the subflow completes. This approach can easily lead to the addition of two compute nodes, each of which performs a tree copy. In such cases the subflow facilitates the reuse of logic but unwittingly adds an additional processing head. Consider the use of ESQL procedures instead for code reuse. They provide a good reuse facility and the potential additional overhead of extra Compute nodes is avoided.

- Existing Java classes can be invoked from within a message flow using either ESQL or Java.
  - From ESQL it is possible to declare an external function or procedure which specifies the name of the Java routine to be invoked.. Parameters (in,out, in-out) can be passed to the Java routine.
  - From within a Java compute node the class can be invoked as any other Java class. You will need to ensure that the Java class file is accessible to the broker runtime for this to work.

- A database stored procedure can be called from within ESQL by using an external procedure. It is also possible to invoke a stored procedure using a PASSTHRU statement but this is not recommended from a performance point of view.

- The cost of calling these external routines will vary and be dependent on what is being called (Java VS Stored procedure) and the number and type of parameters. The performance reports give the CPU cost of calling different procedure types and you can compare the cost of using these methods to other processing which you might do.

- There are no hard and fast rules about whether to call an external procedure or not. It will very much depend on what code you want to access. It is important to bear in mind the cost of making the call against the function that is being executed. Is it really worth it if the external procedure is only a few lines of code. It might easier to re-write the code and make it available locally within the message flow.

- Database stored procedures can be a useful technique for reducing the size of the result set that is returned to a message flow from a database query. Especially where the result set from an SQL statement might contain hundreds or thousands of rows otherwise. The stored procedure can perform an additional refinement of the results. The stored procedure is close to the data and using it to refine results is a more efficient way of reducing the volume of data. Otherwise data has to be returned to the message flow only to be potentially discarded immediately.

Contents

Business Processing

- The choices…
  - ESQL
  - Java
  - Mapping Node
  - eXtensible Stylesheet Language (XSL)
  - WebSphere Transformation Extender (WTX)
  - PHP

- **Factors in choosing a transformation option:**
  - **Existing skills**
  - **Availability of new skills**
  - **Re-use of existing assets**
  - **Throughput requirements**

Business Processing

N O T E S

- There are a number of different transformation technologies available with Message Broker V6. Theses are:
  - ESQL, an SQL like syntax, which runs in nodes such as the Compute, Filter and Database.
  - Java which runs in a JavaCompute node. In releases prior to V6 this would have been a plug-in node.
  - The Mapping node. The mapping capability has been rewritten in V6. It gives a spreadsheet style format in which you can specify the connection of source and destination fields along with some additional processing.
  - Extensible Stylesheets (XSL) running in an XML Transformation node. This provides a good opportunity to reuse an existing asset.
  - WebSphere TX (WTX) mappings using the WebSphere TX Extender node. The users of this are most likely to be those with existing WTX mappings who want to run them in the broker environment.
- As there is a choice of transformation technology this offers significant flexibility. All of these technologies can be used within the same message flow if needed.
- Different technologies will suit different projects and different requirements.
- Factors which you may want to take into account when making a decision about the use of transformation technology are:
  - The pool of development skills which is available. If developers are already skilled in the use of ESQL you wish continue using this as the transformation technology of choice. If you have a strong Java skills then the JavaCompute node may be more relevant, rather than having to teach your Java programmers ESQL.
  - The ease with which you could teach developers a new development language. If you have many developers it may be not be viable to teach all of the ESQL for example and you may only educate a few and have them develop common functions or procedures.
  - The skill of the person. Is the person an established developer, or is the mapping node more suitable.
  - Asset re use. If you are an existing user of Message Broker you may have large amounts of ESQL already coded which you wish to re-use. You may also have any number of resources that you wish to re-use such as Java classes for key business processing, stylesheets or WTX mappings. You want to use these as the core of message flow processing and extend the processing with one of the other technologies.
  - Performance. If message throughput is a prime consideration then you will most likely want to consider using ESQL or Java.

Contents

Transactional Processing

- Transactional support provided by WebSphere MQ
  - Local units of work for message processing
  - Global units of work for message and external data processing
  - Support for two phase commit with JMS client
  - Transactions necessitate log I/O – can change processing from CPU bound to I/O bound
- MQPUT/MQGET out of syncpoint
  - Early availability of messages
  - No back out
  - On distributed platforms use persistent messages in unit of work or in syncpoint
- Message batching
  - Reduces number of commits performed by the message flow
  - Good where many messages to be processed
  - Has an impact on response time and recovery
  - Specify in BAR File > Configure > Commit Count
- If not using provided transactional support
  - What will mange data integrity
  - Is there retry logic

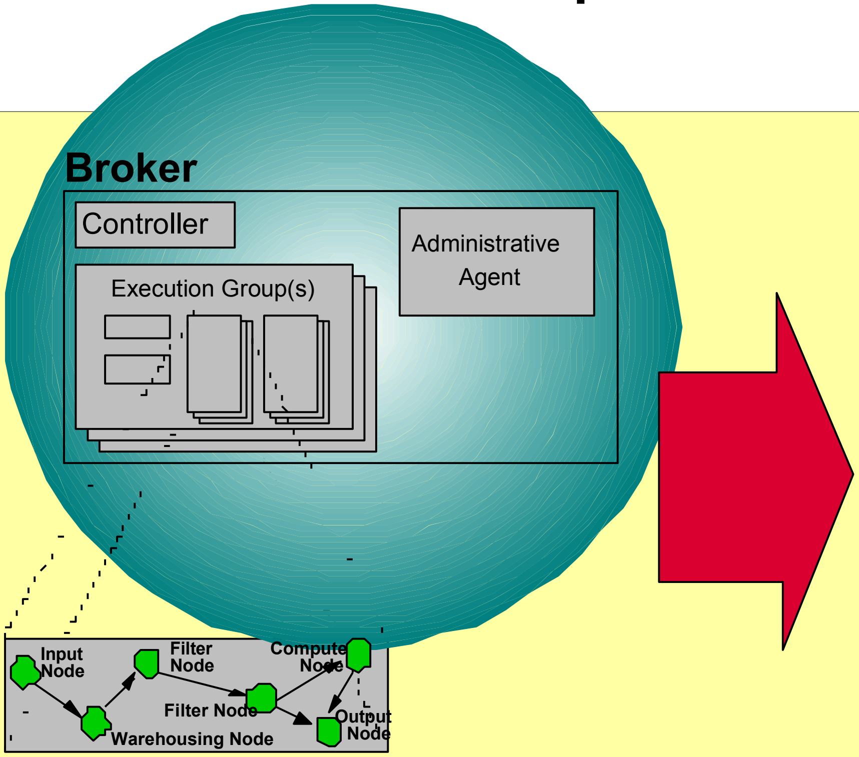
Transactional Processing

N O T E S

- ❑ For most applications it is essential that data is not lost. That is why many companies use WebSphere MQ for messaging. Assured delivery, once and only once is what WebSphere MQ is famous for. If you need this level of protection you must use persistent messages.
- ❑ What ever the requirement it is important to understand what the recovery requirements are from the beginning of the message flow design/architecture phase. Data integrity must be part of the design and not something which you attempt to add at the end of the implementation. Providing integrity requires you to examine the configuration of the broker, its queue manager and any database from which business data is processed as well as the applications which create input for the message flows and consume the output messages.
- ❑ Message Broker offer two levels of data protection. These are local and global units of work. Local units of work are the protection of message operations. This means MQGET and MQPUT operations can take place within a unit of work and all will take place if the unit of work is committed or non if the unit of work is backed out. Message Broker uses the facilities of the WebSphere MQ queue manager for this recovery processing and does not provide its own unit of work processing capability.
- ❑ Individual message operations can form part of a unit of work, so not all operations in a message flow have to be within a unit of work. In order to have message operations performed within a unit of work you must specify a value for transactionMode on the appropriate MQInput or MQOutput nodes in the message flow.
- ❑ Global units of work provide protection for messaging operations and database update/insert activity. That is database operations involving business data. This is data accessed through one of the database nodes, a filter node or a PASSTHRU statement in a compute node.
- ❑ As with Local units of work, brokers use the recovery facilities of WebSphere MQ. In particular the two phase commit support which it provides. For Global units of work to function there must be an XA interface defined between the database manager and the brokers queue manager on the Windows and UNIX platforms. On z/OS RRS is used to perform the transaction coordination role.
- ❑ The whole message flow must operate within a global unit of work. However it is possible for individual nodes to perform work outside the unit of work so that the processing performed in the node is committed unilaterally, regardless of the success of the transactional processing.
- ❑ In order to select global units for a message flow the property coordinatedTransaction must be set for the message flow in the assignments view in the Message Broker Toolkit when the message flows runs on Windows or UNIX. On z/OS the default is to run the message flow with a global unit of work. This is handled automatically by RRS working in conjunction with the resource managers.
- ❑ Be aware that messages which are not processed as part of a unit of work will become visible almost immediately to other applications and will not wait for the completion of the unit of work. This may be an effect that you are after, on the other hand it may be a problem.
- ❑ Message batching is a useful technique for sharing the cost of log writes amongst multiple applications. Multiple messages can be written in the same commit. This is a useful technique to use where there are many persistent messages being written concurrently. Be aware that in the event of a failure multiple messages could be backed out. Normal rules of transactional processing will be observed though. Message batching is controlled at the individual message flow level. To turn it on specify a value for the commit count. This is accessed by using the BAR File editor, choosing the configure tab and then setting commit count on the panel displayed. Save the change and redeploy the message flow for it to take effect.
- ❑ In this section we have discussed the use of transactional support as provided by WebSphere MQ. It does not have to be used. There are alternative strategies such as application compensation, or retry logic when a response is not received to a request. If you do decide to use one of these other techniques it is important to ensure that end to end data integrity is maintained.

Contents

Execution Groups and Additional Instances



Process/Address Spaces

Administrative Agent
Controller
Execution Groups

Thread/TCB

Message Flows

✓ Threading model provided
✓ Great potential for multiprocessor exploitation

Execution Groups and Additional Instances

N O T E S

- ❑ Within a broker it is possible to have one or more execution groups.
- ❑ Execution group is a broker term and is a container in which one or more message flows can run.
- ❑ The implementation of an execution group varies with platform. On Windows and UNIX platforms it is implemented as an operating system process. On z/OS it is implemented as an address space.
- ❑ The execution group provides separation at the operating system level. If you have message flows which you must separate for some reason, you can achieve this by assigning them to different execution groups.
- ❑ An individual message flow runs as an operating system thread on the Windows and UNIX platforms and as a Task Control Block (TCB) on z/OS.
- ❑ It is possible to run more than one copy of a message flow in an execution group, in which case there will be multiple threads or TCBs running the same message flow. Equally you can run a message flow in more than one execution group. In which case there will be one or more threads or TCBs running the message flow in each of the processes or address spaces to which the message flow has been assigned.
- ❑ A significant benefit of using Message Broker is that the threading model is provided as standard. The message flow developer does not need to explicitly provide code in order to cope with the fact that multiple copies of the message flow might be run. How many copies and where they are run is an operational issue and not a development one. This provides significant flexibility.
- ❑ The ability to use multiple threads or TCBs and also to replicate this over multiple processes or address spaces means that there is excellent potential to use and exploit multiprocessor machines.

Maximising the Scalability of Message Flows

- Key Factors
  - Design
    - Minimise use of
      - Resources (Queues, databases)
      - Other shared resources
  - Coding
  - Tuning

(all the material in this presentation!)

Message Flow Characteristics

- *Efficient*
- *CPU bound*
- *Minimum I/O*
- *No affinities*
- *No serialisation*

- Test the application at a high processing volume
- Investigate the different scaling techniques
  - Results will vary by application

Maximising the Scalability of Message Flows

N O T E S

- ❑ Sometimes the message rate that can be obtained from a single copy of a message flow is not sufficient to meet the message processing rate that your business application must provide. As such it will be necessary to run multiple copies of the message flow. There are a number of techniques that you can use to do this. We will discuss these shortly but before we do lets take moment to think about the characteristics of the message flows that you will be running.
- ❑ In order to improve the scalability of your message flows, that is the rate at which throughput increases with additional copies of the message flow running, it is important to make sure you have an efficient design, that it was coded well and efficiently and that the broker and any dependent resources are well tuned. You also want the minimum contention for common resources like a control record in a database.
- ❑ The goal is to have message flows which are CPU bound, perform minimum I/O, have no affinities and no serialised processing in them. Such message flows will scaling more easily and achieve a better processing rate than those which do not have these characteristics.
  - o Message flows which are CPU bound scale more easily than those which are I/O bound. Whilst you can reduce the impact of having to perform I/O by using a SAN instead of SCSI disks for example it is much better to not have to perform the I/O in the first place. Look to see where data wi;ll be help and updated in your proposed design. Does an entry have to be inserted into a database for every incoming message for example or could you save that data to a persistent MQ message and insert in the database at a later time. As an illustration of the difference that minimising I/O can make take the example of the Aggregation nodes in WebSphere Message Broker V2 and V5 compared with the implementation in later editions of 5 and V6 onwards. In that original implementation status information was stored in a database. For every message that was fanned-out and each reply message received a row had to be inserted into a database. This made the database log a point of contention. Dependent on the speed of the device that the log was located on your would get a different message rate. When the Aggregation nodes where modified to use MQ queues there was no longer a dependency on the database log and as a result there were improvements in performance of up to 10 times on some platforms. So there you have the same function implemented differently and a significant difference in the performance characteristics.
  - o Serialised processing is a major inhibitor to achieving high processing volumes and should be avoided at all costs. In this presentation we look a couple of suggestions to help minimise the impact of it.
  - o It is important to avoid affinities to particular instances of an application or to a particular machine if you want to increase message throughput otherwise you risk becoming limited by the capacity of one application or machine.
- ❑ It is important that you test your message flows with a high volume of processing. Without doing this you cannot know at what point processing will become bottlenecked. Aim to test to two times, three times or more your required throughput so that you have contingency. There are a number of tools to help you in this testing, some of the free (see the section on Tools later in the presentation).
- ❑ There a number of different techniques that you can use to run more copies of a message flow. Try several of them to see which works best for each application. What is the best technique for one may not necessary be the best for the others.

Increasing Message Throughput

- Additional Instances
  - Broker supported mechanism in which you specify a thread pool for the message flow or input node
  - Results in additional thread(s)
  - ***Cheapest way to increase throughput***
- Copying message flows
  - Copy a message flow and rename
  - Can present code management issues as you now have multiple copies of flow
  - Results in additional thread(s)
  - Cheap to implement
- Execution Groups
  - Offers process level separation
  - Typically requires 150MB+ a time
- Multiple Brokers
  - Most expensive to implement and resource
  - Only recommended when capacity of the queue manager is exhausted (log capacity for example) or there is a need for complete separation of projects

Message Flow Deployment Algorithm

Execution Groups

Many or few ?

Additional Instances

In one execution group or many?

Execution Groups

- Results in new process/address space
- High(er) memory requirement
- Multiple threads incl. management
- Operational simplicity
- Gives Process level separation

Additional Instances

- Results in one or more threads
- Low(er) memory requirement
- Thread allocation issues pre V6.1
- Thread level separation
- Potential to use shared variables

- **Recommended Usage**
  - One (or two) execution groups per application
  - Allocate all of the instances needed for a flow over those execution groups
  - Assign heavy resource users (memory) to specialist execution groups

- **Ultimately have to balance: Resource use, Manageability & Availability**

Message Flow Deployment Algorithm

NOTES

- In testing and production you will need to decide on the which message flows are assigned to which execution groups and how many execution groups are allocated. There are a number of possible approaches: Allocate all of the message flows in to one or two execution groups: Have an execution group for each message flow; Split message flows by project and so on. Before considering what the best approach is lets look at some of the characteristics of execution groups and additional instances.
- Execution Group:
  - Each execution group is an operating system process or address space in which one or more message flows run. Each new execution group will result in an additional process or address space being run.
  - An execution group might typically require 150MB of memory to start and load executables. Its size after initialisation will then depend on the requirements of the message flows which run within it. Each additional execution group which is allocated will need a minimum of another 150MB so you can see that using a high number of execution groups is likely to use a large amount of memory quickly.
  - As well as the threads needed to run a message flow an execution group also has a number of additional threads which are use to perform broker management functions. So allocating many execution groups will result in more management threads being allocated overall.
  - An execution group provides process level separation between different applications. This can be useful for separating different applications.
- Additional Instance:
  - Each new additional instance results in one more thread being allocated in an existing execution group. As such the overhead is low. There will be some additional memory requirements but this will be significantly less than the 150MB that is typically needed to accommodate a new execution group.
  - The way in which additional instances are used in a message with multiple input nodes is not deterministic. This could lead to thread starvation for some input nodes in certain situations. This can now be avoided in Message Broker V6.1.
  - Message flows running as instances in a single execution group can access common variables called shared variables. This gives a cheap way to access low volumes of common data. The same is not true of messages flows assigned to different execution groups.
- When assigning message flows to execution groups there is no fixed algorithm imposed by Message Broker. There is significant flexibility in the way in which the assignment can be managed. Some users allocate high numbers of execution groups and assign a few message flows to each. We have seen several instance of 100+ execution groups being used. Others allocate many message flows to a small number of execution groups.
- A key factor in deciding how many execution groups to allocate is going to be how much free memory you have available on the broker machine.
- A common allocation pattern is to have one or possibly two execution groups per each project and to then assign all of the message flows for that project to those execution groups. By using two execution groups you build in some additional availability should one execution fail.
- Where you have message flows which require large amounts of memory in order to execute, perhaps because the input messages are very large or have many elements to them, then you are recommended to keep all instances of those message flows to a small number of execution groups rather than allocate over a large number of execution groups [It is not uncommon for an execution group to use 1GB+ of memory.]..Otherwise every execution group could require a large amount of memory as soon as one of the large messages was processed by a flow in that execution group. Total memory usage would rise significantly in this case and could be many gigabytes.
- As with most things this is about achieving a balance between the flexibility you need against the level of resources (and so cost) which have to be assigned to achieve the required processing.

How Many Instances of a Message Flow?

- The 3 rules:

#1 – there is no pre-set magic number

#2 – flows are like people – different and sensitive.

  - What are they sensitive to?

#3 – you need to run it for real to find out what is needed

- Some key questions
  - What message rate do you require for the application?
  - What rate will one instance of the message flow(s) sustain?
  - How well does the message flow scale?
    - **CPU VS I/O bound**
    - **Contention e.g. database, access to a single queue**

- Solutions using multiple message flows will require different numbers of running copies for each part to achieve balance

How Many Instances of a Message Flow to Use?

N O T E S

- A common question when configuring a production system is how many instances of a particular message flow should be used. This is not possible to predict ahead of time and can only be determined by undertaking some performance testing to see what level of message throughput can be obtained with different number of instances of the message flow.

- There is no pre-set magic number on the number of instances to use any situation. We cannot say if there is an MQInput you need 5 instances. If you have 3 compute nodes then you need 6 instances etc. Every message flow will be different. The only way to determine what is the right number of instances in any situation is to run the message flow and conduct a performance test. We discuss a number of free tools later in the presentation which could be used for this type of testing.

- If we step back from the question for a moment. What is it that we want to achieve with additional instances? What we really want to do is to be able to obtain a certain level of message throughput for a particular message flow or group of flows.

- With any message flow we will achieve a particular message rate when it is run. Exactly what this is depends on the processing characteristics of the flow and the resources allocated to running it. Message throughput can normally be increased by running more copies of the message flow and that is what additional instances is all about.
- Instead of using additional instances another approach would be to assign a copy of the message flow to its own execution group. This would increase the total number of copies of the flow which are running but uses significantly more memory to do it than using additional instances in a small number of execution groups which is why the focus tends to be on using additional instances first.

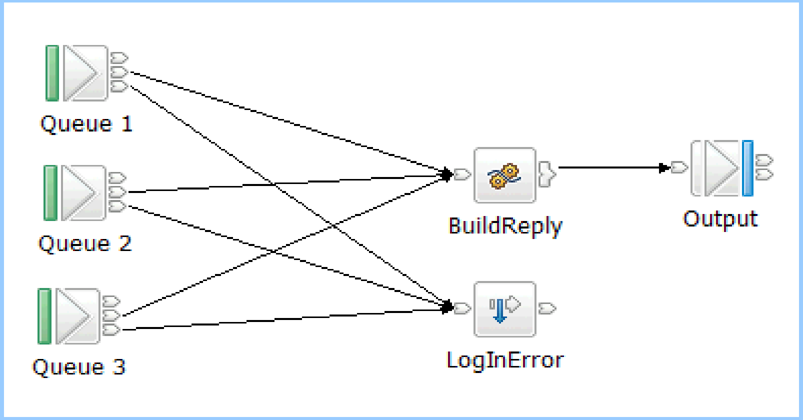
- Before we can allocate a number of additional instances we need to know what the target message rate is for the application, this is driven by business requirements. We then simply increase the number of copies of the message flow which run so that we can achieve the required rate. When you do this plot the message rate against the number of copies of the flow running to observe the effectiveness of more additional instances. If message rate peaks in one execution group then you are recommended to run a second, third fourth execution group etc. with the same number of instances as the first and look at the effectiveness of that.

- The effectiveness of increasing the number of instances for a message flow will depend on the processing characteristics of the message flow. A message flow which is CPU bound will increase message throughput more rapidly than one which is I/O bound or which has some contention in it.

- Where you have a group of message flows which work together to support an application you may well find that different numbers of instances are needed for of each of the individual message flows. One size may well not fit all. This is very common when using the aggregation nodes for example. The number of instances needed for the fan-out processing will be different for those needed for the fan-in processing.

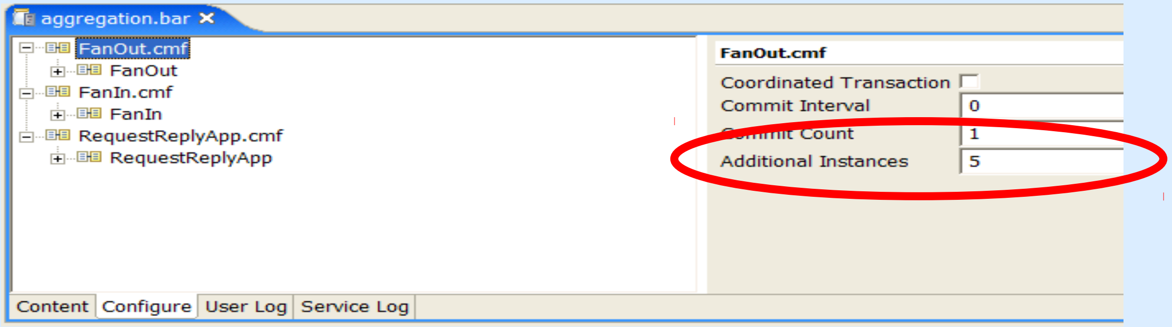
Additional Instances

- Prior to V6.1 thread usage is not deterministic with multiple input nodes and additional instances specified
  - Have more than one consumer of threads
  - Thread starvation as input node for deepest queue consumes most threads/instances



Additional Instances

- Prior to V6.1 specified in the BAR file only



aggregation.bar

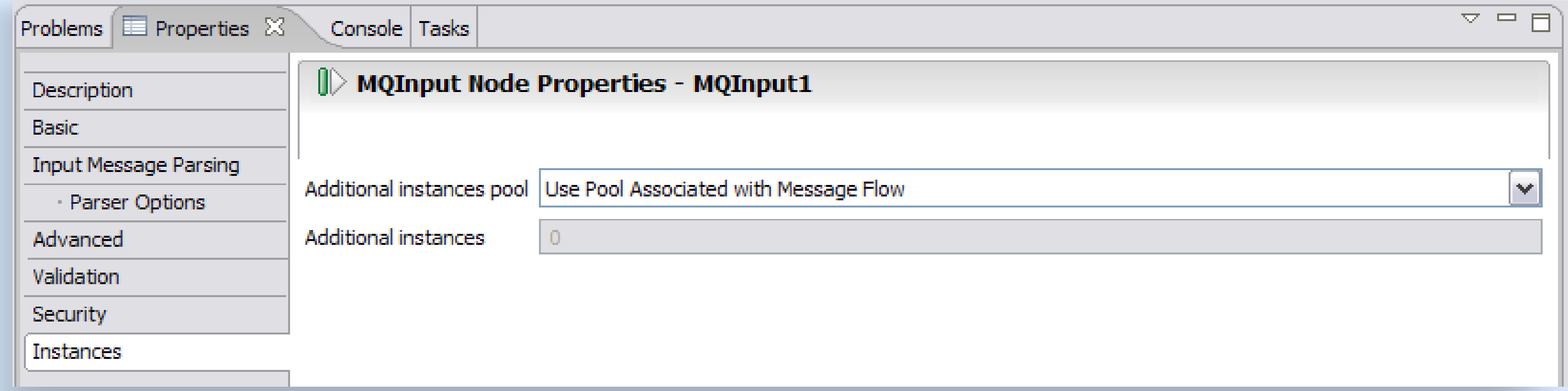
- FanOut.cmf
  - FanOut
- FanIn.cmf
  - FanIn
- RequestReplyApp.cmf
  - RequestReplyApp

FanOut.cmf

Coordinated Transaction	☐
Commit Interval	0
Commit Count	1
Additional Instances	5

Content | Configure | User Log | Service Log

- Now also available at the MQInput node level
- Prevents possible thread starvation when using multiple input nodes in a single flow



Problems | Properties | Console | Tasks

MQInput Node Properties - MQInput1

Description, Basic, Input Message Parsing, Parser Options, Advanced, Validation, Security, Instances

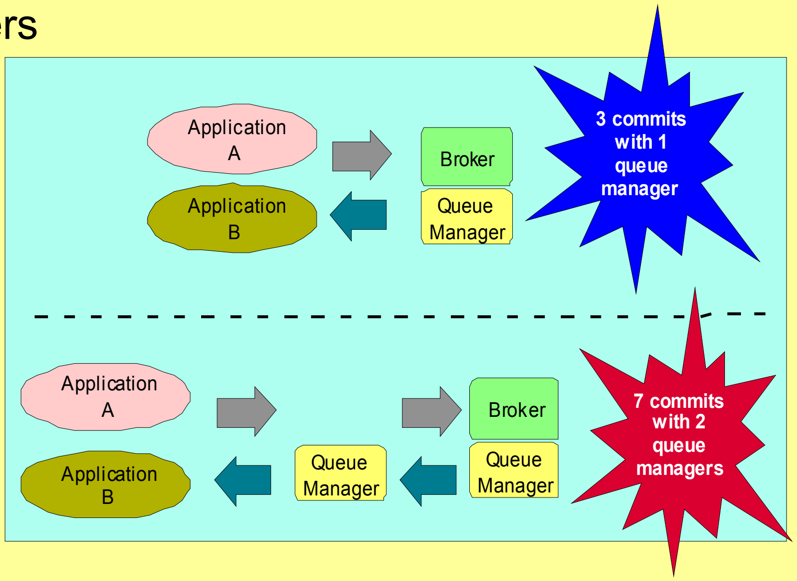
Additional instances pool	Use Pool Associated with Message Flow
Additional instances	0

Contents

Tuning the runtime environment

- Broker
  - Trusted application or not (not AIX with MQ < V6.0 or z/OS (where it is the default))
  - Use of instances and execution groups
  - Non MQ Transport (HTTP, JMS)
  - Java Heap size
- Broker queue manager
  - Logging
  - Trusted channels and listeners
  - Performance of connected queue managers
- Broker and Application Databases
  - Review activity and tune
- Topology
  - Distance between broker and data

See developerWorks articles for MQ and HTTP tuning advice



Tuning the runtime environment

N O T E S

- So far we have focused on the message and contents of the message flow. It is important to have an efficient message flow but this is not the whole story. It is also important to have an efficient environment for the message flow to run in.

- We collectively refer to this as the broker environment. It includes:
  - Configuration of the broker
  - Configuration of the broker queue manager,
  - The topology of the environment in which the messages are processed
  - Broker database configuration
  - Business database configuration
  - Hardware
  - Software

- It is important that each of these components is well configured and tuned. When this occurs we can get the best possible message throughput for the resources which have been allocated.

- We will now briefly look at each of these aspects. The aim is to outline those aspects which you should be aware of and further investigate rather than to provide full guidance in this presentation.

Tuning the runtime environment

N O T E S

- As with other MQ applications it is possible to run a broker as a trusted application on the Windows NT, Windows 2000, Solaris HP-UX platforms. AIX is also supported but only when queue manager and Message Broker are running in 64 bit mode. The benefit of running in trusted mode is that the cost of an application communicating with the queue manager is reduced and MQ operations are processed more efficiently. It is thus possible to achieve a higher level of messaging with a trusted application. However when running in trusted mode it is possible for a poorly written application to crash and potentially corrupt the queue manager. The likelihood of this happening will depend on user code, such as a plugin node, running in the message flow and the extent to which it has been fully tested. Some users are happy to accept the risk in return for the performance benefits. Others are not.

- Using the Message Broker Toolkit it is possible to increase the number of instances of a message flow which are running. Similarly you can increase the number of execution groups and assign message flows to those execution groups. This gives the potential to increase message throughput as there are more copies of the message flow. There is no hard and fast rule as to how many copies of a message flow to run. For guidelines look at the details in the Additional Information section

- Tuning is available for non MQ transports. For example:
  - When using HTTP you can tune the number of threads in the HTTP listener for example. For more detail on how to tune the HTTP support within Message Broker see the article HTTP transport nodes in WebSphere Message Broker V6 at

  http://www.ibm.com/developerworks/websphere/library/techarticles/0608_braithwaite/0608_braithwaite.html
  - When using JMS nodes follow the tuning advice for the JMS provider that you are connecting to.

- As the broker is dependent on the broker queue manager to get and put MQ messages the performance of this component is an important component of overall performance.

- When using persistent messages it is important to ensure the queue manager log is efficient. Review log buffer settings and the speed of the device on which the queue manager log is located.

- It is possible to run the WebSphere queue manager channels and listener as trusted applications. This can help reduce CPU consumption and so allow greater throughput.

- In practice the broker queue manager is likely to be connected to other queue managers. It is important to examine the configuration of these other queue managers as well in order to ensure that they are well tuned.

Tuning the runtime environment

N O T E S

- ❑ The input messages for a message flow do not magically appear on the input queue and disappear once they have been written to the output queue by the message flow. The messages must some how be transported to the broker input queue and moved away from the output queue to a consuming application.
- ❑ Let us look at the processing involved to process a message in a message flow passing between two applications with different queue manager configurations.
  - o Where the communicating applications are local to the broker, that this they use the same queue manager as the broker, processing is optimized. Messages do not have to move over a queue manager to queue manager channel. When messages are non persistent no commit processing is required. When messages are persistent 3 commits are required in order to pass one message between the two applications. This involves a single queue manager.
  - o When the communicating applications are remote from the broker queue manager, messages passed between the applications must now travel over two queue manager to queue manager channels (one on the outward, another on the return). When messages are non persistent no commit processing is required. When messages are persistent 7 commits are required in order to pass one message between the two applications. This involves two queue managers.
- ❑ If the number of queue managers between the applications and broker where to increase so would the number of commits required to process a message. Processing would also become dependent on a greater number of queue managers.
- ❑ Where possible keep the applications and broker as close as possible in order to reduce the overall overhead.
- ❑ Where multiple queue managers are involved in the processing of messages it is important to make sure that all are optimally configured and tuned.

Tuning the runtime environment

N O T E S

- The extent to which the broker database will be used depends on which facilities are used in the message flows. Use of publication nodes can lead to increased use of the broker database. In this case it will be necessary to ensure that the broker database is well tuned, especially to handle insert/delete activity. The performance of the database log will then become important and may be a gating factor in overall performance.

- The extent to which business data is used will be entirely dependent on the business requirements of the message flow. With simple routing flows it is unlikely that there will be any database access. With complex transformations there might be involved database processing. This could involve a mixture of read, update, insert and delete activity.

- It is important to ensure that the database manager is well tuned. Knowing the type of activity which is being issued against a database can help when tuning the database manager as you can focus tuning on the particular type of activity which is used.

- Where there is update, insert or delete activity the performance of the database log will be a factor in overall performance. Where there is a large amount of read activity it is important to review buffer allocations and the use of table indices for example.

- It might be appropriate to review the design of the database tables which are accessed from within a message flow. It may be possible to combine tables and so reduce the number of reads which are made for example. All the normal database design rules should be considered.

- Where a message flow only reads data from a table, consider using a read only view of that table. This reduces the amount of locking within the database manager and reduces the processing cost of the read.

Broker Environment - Hardware & Software

- Understand message processing requirements
  - Run the message flows with realistic data variety of types & sizes
  - Use early testing to plan/validate production environment

- I/O or CPU bound
  - Depends on msg type, size and complexity of flows
  - Use vmstat, Task Manager, SDSF etc to determine which

- Number and speed of processors

- Software
  - Ensure it is tuned
  - Use latest levels

Broker Environment - Hardware & Software

N O T E S

- ❑ Allocating the correct resources to the broker is a very important implementation step.
- ❑ Starve a CPU bound message flow of CPU and you simply will not get the throughput. Similarly neglect I/O configuration and processing could be significantly delayed by logging activities for example.
- ❑ It is important to understand the needs of the message flows which you have created. Determine whether they are CPU bound or I/O bound.
- ❑ Knowing whether a message flow is CPU bound or I/O bound is key to knowing how to increase message throughput. It is no good adding processors to a system which is I/O bound. It is not going to increase message throughput. Using disks which are much faster such as a solid state disk or SAN with a fast-write non-volatile cache will have a beneficial effect though.
- ❑ As we saw at the beginning Message Broker has the potential to use multiple processors by running multiple message flows as well as multiple copies of those message flows. By understanding the characteristics of the message flow it is possible to make the best of that potential.
- ❑ In many message flows parsing and manipulation of messages is common. This is CPU intensive activity. As such message throughput is sensitive to processor speed. Brokers will benefit from the use of faster processors. As a general rule it would be better to allocate fewer faster processors than many slower processors.
- ❑ We have talked about ensuring that the broker and its associated components are well tuned and it is important to emphasize this. The default settings are not optimized for any particular configuration. They are designed to allow the product to function not to perform to its peak.
- ❑ It is important to ensure that software levels are as current as possible. The latest levels of software such as Message Broker and WebSphere MQ offer the best levels of performance.

Contents

Common Problems

- Parsing
  - Excessive
  - Expensive
- Poor transformation logic
  - Avoiding most common problems
- Trace
  - Trace Nodes in flows (pre V6.1)
  - Product trace turned on
- Runtime configuration
  - Instances/Execution Groups
  - Message batching
- Transport tuning (see performance reports)
  - HTTP
  - MQ
  - JMS
  - TCP/IP and network
- Transactional Processing
  - Queue Manager Logs
  - Inconsistent use of txnl processing
- Struggling to find the problem
  - Knowing how to look inside the flow
- Test methodology
  - Preloading queues VS top-up
  - Separation of client and broker

Common Problems

N
O
T
E
S

- The purpose of this section is to highlight the most common performance related problems so that you can hopefully avoid them.
- Coding tips follow for ESQL, Java and XSLT in next four pages of notes.
- With product trace check to ensure that trace is not turned on against specific message flows or against the execution. Check even if you think you turned trace off.
- From Message Broker V6.1 the overhead of having trace nodes wired in message flows is minimal. Previously there was a significant overhead.
- Follow the discussion in earlier foils about the use of multiple execution groups and additional instances of message flows.
- Be sure to follow transport specific tuning hints and tips.
- If message flows use persistent MQ messages be sure to review the configuration and placement of the broker queue manager logs and all interconnected queue managers which will process messages on both the inbound and outbound sides.
- The results which you obtain when conducting performance tests can in part be determined by the way in which you conduct the tests. It is always better to measure sustained throughout using a tool which can continuously send and receive messages rather than pre-loading a batch of messages on to the input queue. Most queue based systems work more efficiently with empty or low queue depths. This is true with both point to point and publish/subscribe processing. Be sure to obtain the tuning advice for the transport and distribution engine that you are using.

Tips for Coding Efficiently in ESQL - 1

NOTES

- Avoid use of array subscripts [ ] on the right hand side of expressions – use reference variables instead. See below. Note also the use of LASTMOVE function to control loop execution.

```
DECLARE myref REFERENCE TO OutputRoot.XML.Invoice.Purchases.Item[1];
-- Continue processing for each item in the array
WHILE LASTMOVE(myref)=TRUE DO
-- Add 1 to each item in the array
SET myref = myref + 1;
-- Move the dynamic reference to the next item in the array
MOVE myref NEXTSIBLING;
END WHILE;
```

- Use reference variables rather than long correlation names such as InputRoot.MRM.A.B.C.D.E. Find a part of the correlation name that is used frequently so that you get maximum reuse in the ESQL. You may end up using multiple different reference variables throughout the ESQL. You do not want to reference variable pointing to InputRoot for example. It needs to be deeper than that.

```
DECLARE myref REFERENCE TO InputRoot.MRM.A.B.C;
SET VALUE = myRef.D.E;
```

- Avoid use of EVAL, it is evil (very expensive!)
- Avoid use of CARDINALITY in a loop e.g. `while ( I < CARDINALITY (InputRoot.MRM.A.B.C[]))`. This can be a problem with large arrays where the cost of evaluating CARDINALITY is expensive and as the array is large we also iterate around the loop more often. See use of reference variables and LASTMOVE above.
- If just checking for the presence of children in an array use EXITS rather than CARDINALITY if possible. EXISTS will stop after the first child, whereas CARDINALITY will count all of them.
- Combine ESQL into the minimum number of compute nodes possible. Fewer compute nodes means less tree copying.
- Use new FORMAT clause (in CAST function) where possible to perform data and time formatting.
- Limit use of shared variables to a small number of entires (tens of entries) when using an array of ROW variables or order in probability of usage (current implementation is not indexed so performance can degrade with higher numbers of entries).
- For efficient code re-use consider using ESQL modules and schema rather than subflows. What often happens with subflows is that people add extra compute nodes to perform initialisation and finalisation for the processing that is done in the subflow. The extra compute nodes result in extra message tree copying which is relatively expensive as it is a copy of a structured object.

Tips for Coding Efficiently in ESQL - 2

NOTES

- ❑ Avoid use of PASSTHRU with a CALL statement to invoke a stored procedure. Use the "CREATE PROCEDURE ... EXTERNAL ..." and "CALL ..." commands instead
- ❑ If you do have to use PASSTHRU use host variables (parameter markers) when using the statement. This allows the dynamic SQL statement to be reused within DB2 [assuming statement caching is active]
- ❑ Declare and Initialize ESQL variables in same statement
  - Example:

```
Declare InMessageFmt     CHAR;
Set InMessageFmt      = 'SWIFT';
DECLARE C INTEGER;
Set C = CARDINALITY(InputRoot.*[]);
```

  - Better to use:

```
Declare InMessageFmt     CHAR ‘SWIFT’;
DECLARE C INTEGER CARDINALITY(InputRoot.*[])
```

- ❑ Initialize MRM output message fields using MRM null/default rather than ESQL
  - o Messageset.mset > CWF>Policy for missing elements
    - Use null value
  - o Cobol import
    - Create null values for all fields if initial values not set in Cobol
  - o Avoids statements like
    - Set OutputRoot.MRM.THUVTILL.IDTRANSIN.NRTRANS   = '';
- ❑ For frequently used complex structures consider pre-building and storing in a ROW variable which is also a SHARED variable (and so accessible by multiple copies of the message flow). The pre-built structure can be copied across during the processing of each message so saving processing time in creating the structure from scratch eace time. For an example of ROW and SHARED variable use see the message flow `Routing_using_memory_cache` which is part of the Message Routing sample in the WebSphere Message Broker Toolkit samples gallery.

Tips for Coding Efficiently in Java

- ❑ Follow applicable points from ESQL like reducing the number of compute nodes
- ❑ Store intermediate references when building / navigating trees

Instead of

```
MbMessage newEnv = new MbMessage(env);
newEnv.getRootElement().createElementAsFirstChild(MbElement.TYPE_NAME,"Destination", null);
newEnv.getRootElement().getFirstChild().createElementAsFirstChild(MbElement.TYPE_NAME, "MQDestinationList", null);
newEnv.getRootElement().getFirstChild().getFirstChild()
createElementAsFirstChild(MbElement.TYPE_NAME,"DestinationData", null);
```

Store references as follows:

```
MbMessage newEnv = new MbMessage(env);
MbElement destination = newEnv.getRootElement().createElementAsFirstChild(MbElement.TYPE_NAME,"Destination", null);
MbElement mqDestinationList = destination.createElementAsFirstChild(MbElement.TYPE_NAME, "MQDestinationList", null);
mqDestinationList.createElementAsFirstChild(MbElement.TYPE_NAME,"DestinationData", null);
```

- ❑ Avoid concatenating java.lang.String objects as this is very expensive since it (internally) involves creating a new String object for each concatenation. It is better to use the StringBuffer class:

Instead of:

```
keyforCache = hostSystem + CommonFunctions.separator + sourceQueueValue +
CommonFunctions.separator + smiKey + CommonFunctions.separator + newElement;
```

Use this:

```
StringBuffer keyforCacheBuf = new StringBuffer();
keyforCacheBuf.append(hostSystem);
keyforCacheBuf.append(CommonFunctions.separator);
keyforCacheBuf.append(sourceQueueValue);
keyforCacheBuf.append(CommonFunctions.separator);
keyforCacheBuf.append(smiKey);
keyforCacheBuf.append(CommonFunctions.separator);
keyforCacheBuf.append(newElement);
keyforCache = keyforCacheBuf.toString();
```

N O T E S

Tips for Coding XPath Efficiently

N O T E S

- Although XPath is a powerful statement from a development point of view it can have an unforeseen performance impact in that it will force a full parse of the input message. By default it will retrieve all instances of the search argument. If you know there is only one element or you only want the first then there is an optimization to allow this to happen. The benefit of using this is that it can save a lot of additional, unnecessary parsing.

- The XPath evaluation in Message Broker is written specifically for the product and has a couple of performance optimizations that are worth noting.

- XPath performance tips:
  - Use /aaa[1] if you just want the first one
    - It will stop searching when it finds it
  - Avoid descendant-or-self (//) if possible
    - It traverses (and parses) the whole message
    - Use /descendant::aaa[1] instead of (//aaa)[1]

Contents

Tools

N O T E S

- In this section we will look at three tools which can significantly help you in operating and performance testing Message Broker.

- The first tool is the WebSphere Message Broker Administration Explorer. The tool is shipped as SupportPac IS02. It comes with full development support. There provides function in two key areas:
  - Administration of the broker including discovery of components and the ability to deploy a BAR File to multiple execution groups.
  - Real-time display of flow performance through the charting of Message Broker Accounting and Statistics data. In the following foils there are examples of data being presented as graphs. It is possible to display information at the flow, thread and node level.

- The second tool is a modular tool that can be used for the testing of JMS and MQ based applications. The tool provides
  - Throttled operation (fixed rate or number of messages)
  - Multiple destinations
  - Live performance reporting
  - JNDI
  - Multiple vendor plug-ins

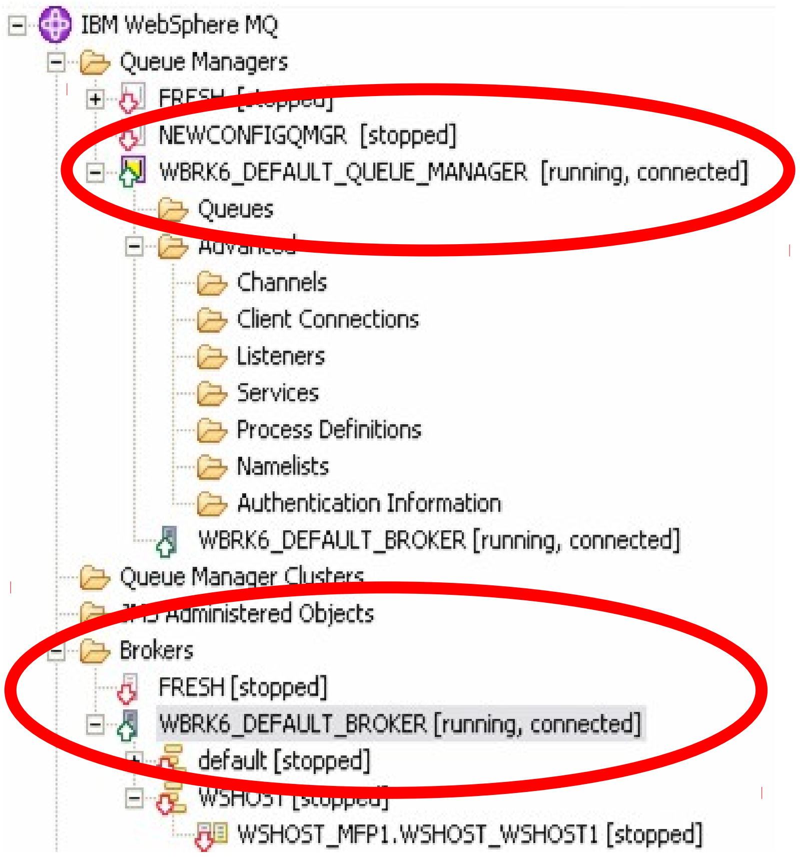
  The URL to download is given on the foil. This is the same tool that the WebSphere Message Broker and WebSphere MQ performance teams use when measuring the product.

- The third tool is the Message display, test & performance utilities. This is shipped as SupportPac IH03. The tool consists of a Windows tools which provides the following facilities:
  - **Capture, store, send test messages**
  - **Display messages in a variety of formats**
  - **Ability to add RFH2 headers**
  - **Manually test PubSub**

  Measure runtime performance using a command line tool. Includes ability to add messages at a rate or flat-out. The data for the messages can be from one or more files.

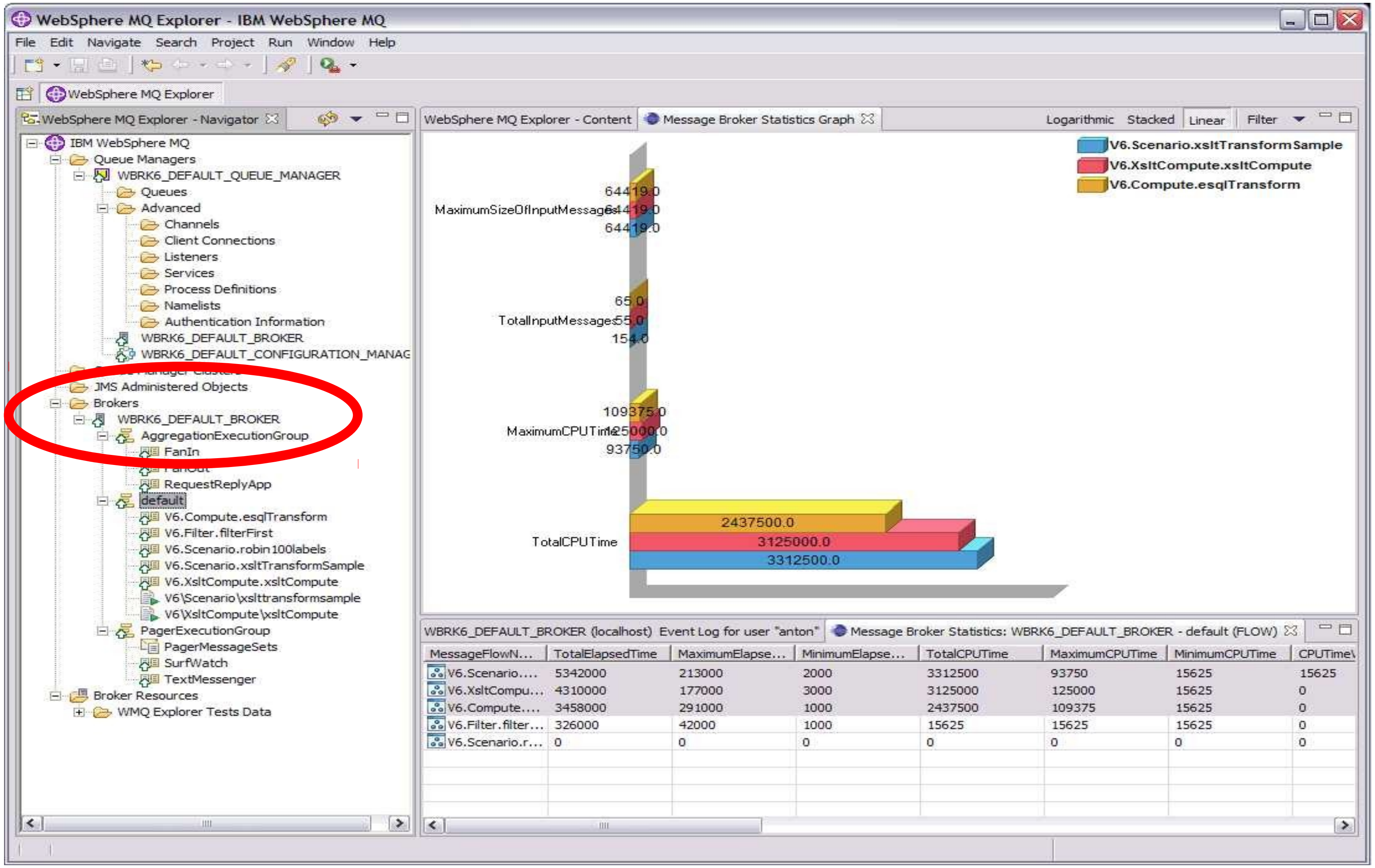
WebSphere Message Broker Explorer

- Administer Message Broker from MQ Explorer
- Broker Explorer current features
  - Automatic discovery of local Brokers
  - Create, Delete, Start and Stop
    - Local Brokers
    - Execution Groups
    - Flows
  - Multiple deploy of BARs to Execution Groups
  - Shipped with Complete Message Broker Documentation
  - Real-time display of flow performance



```
IBM WebSphere MQ
  Queue Managers
    FRESH [stopped]
    NEWCONFIGQMGR [stopped]
    WBRK6_DEFAULT_QUEUE_MANAGER [running, connected]
      Queues
      Advanced
        Channels
        Client Connections
        Listeners
        Services
        Process Definitions
        Namelists
        Authentication Information
      WBRK6_DEFAULT_BROKER [running, connected]
  Queue Manager Clusters
  JMS Administered Objects
  Brokers
    FRESH [stopped]
    WBRK6_DEFAULT_BROKER [running, connected]
      default [stopped]
      WSHOST [stopped]
        WSHOST_MFP1.WSHOST_WSHOST1 [stopped]
```

Accounting and Statistics Flow level data


WebSphere MQ Explorer - IBM WebSphere MQ
File Edit Navigate Search Project Run Window Help
WebSphere MQ Explorer
WebSphere MQ Explorer - Navigator
IBM WebSphere MQ
Queue Managers
WBRK6_DEFAULT_QUEUE_MANAGER
Queues
Advanced
Channels
Client Connections
Listeners
Services
Process Definitions
Namelists
Authentication Information
WBRK6_DEFAULT_BROKER
WBRK6_DEFAULT_CONFIGURATION_MANAG
JMS Administered Objects
Brokers
WBRK6_DEFAULT_BROKER
AggregationExecutionGroup
FanIn
RequestReplyApp
default
V6.Compute.esqlTransform
V6.Filter.filterFirst
V6.Scenario.robin100labels
V6.Scenario.xsltTransformSample
V6.XsltCompute.xsltCompute
V6\Scenario\xslttransformsample
V6\XsltCompute\xsltCompute
PagerExecutionGroup
PagerMessageSets
SurfWatch
TextMessenger
Broker Resources
WMQ Explorer Tests Data
WebSphere MQ Explorer - Content
Message Broker Statistics Graph
Logarithmic Stacked Linear Filter
V6.Scenario.xsltTransformSample
V6.XsltCompute.xsltCompute
V6.Compute.esqlTransform
MaximumSizeOfInputMessages
64419.0
64419.0
64419.0
TotalInputMessages
65.0
55.0
154.0
MaximumCPUTime
109375.0
125000.0
93750.0
TotalCPUTime
2437500.0
3125000.0
3312500.0
WBRK6_DEFAULT_BROKER (localhost) Event Log for user "anton"
Message Broker Statistics: WBRK6_DEFAULT_BROKER - default (FLOW)

MessageFlowN...	TotalElapsedTime	MaximumElapse...	MinimumElapse...	TotalCPUTime	MaximumCPUTime	MinimumCPUTime	CPUTime\
V6.Scenario....	5342000	213000	2000	3312500	93750	15625	15625
V6.XsltCompu...	4310000	177000	3000	3125000	125000	15625	0
V6.Compute....	3458000	291000	1000	2437500	109375	15625	0
V6.Filter.filter...	326000	42000	1000	15625	15625	15625	0
V6.Scenario.r...	0	0	0	0	0	0	0

JMSPerfHarness

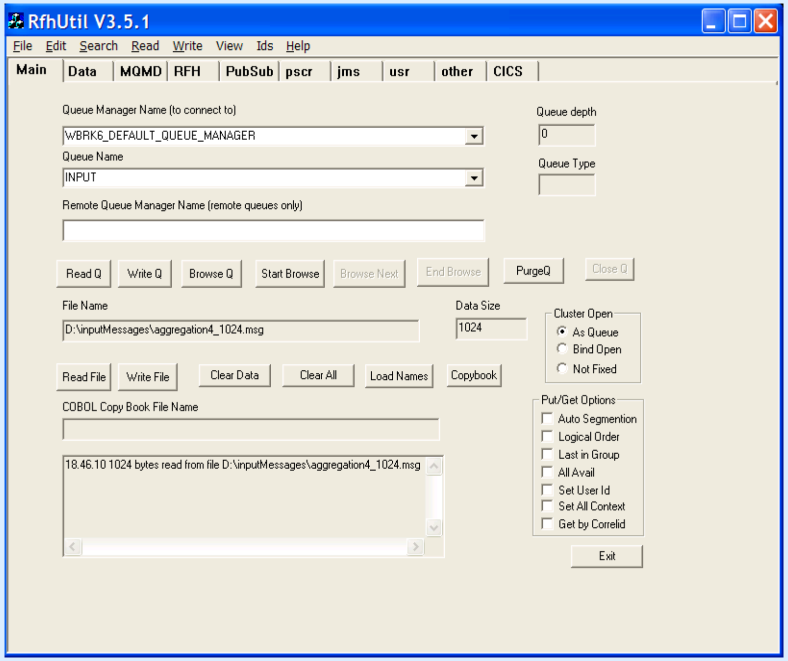
- Modular tool for the testing of JMS and MQ based applications
- Enables easy testing of JMS providers & other messaging apps
- Provides
  - Throttled operation (fixed rate or number of messages)
  - Multiple destinations
  - Live performance reporting
  - JNDI
  - Multiple vendor plug-ins
- Used by WebSphere Message Broker & WebSphere MQ performance teams
- Available at http://alphaworks.ibm.com/tech/perfharness

Message display, test & performance utilities

- A Windows based tool
  - Capture, store, send test messages
  - Display messages in variety of formats
    - Character, Hex, Both, XML, Parsed COBOL
    - ASCII, EBCDIC
    - Simple Chinese, Traditional Chinese,
    - Korean, Japanese
  - Add RFH2 headers
  - Manually test PubSub

- Measure runtime performance
  - Command line tool
  - Put messages at a rate or flat-out
  - Read from one file or multiple
  - Measure output message rate

- Shipped as SupportPac IH03


RfhUtil V3.5.1
File Edit Search Read Write View Ids Help
Main Data MQMD RFH PubSub pscr jms usr other CICS
Read Q Write Q Browse Q Start Browse Browse Next End Browse PurgeQ Close Q
Read File Write File Clear Data Clear All Load Names Copybook
Exit

Queue Manager Name (to connect to): WBRK6_DEFAULT_QUEUE_MANAGER
Queue depth: 0
Queue Name: INPUT
Queue Type:
Remote Queue Manager Name (remote queues only):
File Name: D:\inputMessages\aggregation4_1024.msg
Data Size: 1024
Cluster Open: [✓ As Queue] Bind Open Not Fixed
COBOL Copy Book File Name:
Put/Get Options: ☐ Auto Segmention ☐ Logical Order ☐ Last in Group ☐ All Avail ☐ Set User Id ☐ Set All Context ☐ Get by Correlid

```
18.46.10 1024 bytes read from file D:\inputMessages\aggregation4_1024.msg
```

Contents

WebSphere Message Broker v7.0 Performance

- Focus areas
  - Administration operations
  - Auditing and monitoring
  - New nodes
  - Performance problem diagnosis via MB Explorer
  - Memory reduction and compiler upgrades
  - VSCR on z/OS

Memory Reduction and Compiler Upgrades

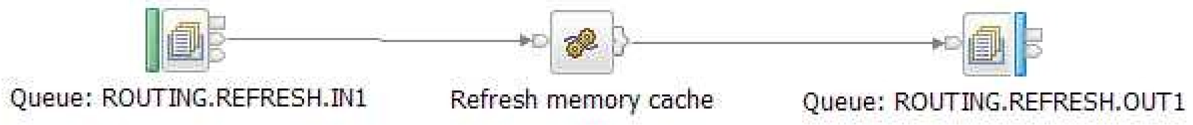
Reduction in Memory Overhead per execution group

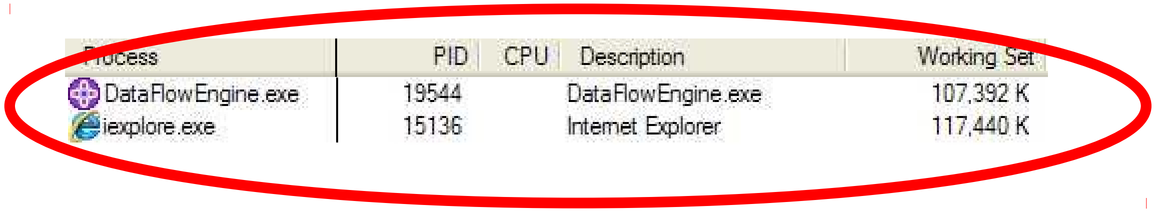
Savings vary by platforms (10-15%)

Compiler upgrades on most platforms → some gains

Product streamlined – no more ConfigMgr or DB

(actually smaller than a web browser!)





Process	PID	CPU	Description	Working Set
DataFlowEngine.exe	19544		DataFlowEngine.exe	107,392 K
iexplore.exe	15136		Internet Explorer	117,440 K

VSCR for z/OS users

Previously users had only ~1.5 GB of private for Message flows with 31 bit support

Compare with 4 GB for 32 bit UNIX systems

Source of frustration for many z/OS customers

MB V7 brings 64 bit support to z/OS

Pre-reqs MQ V7.0.1 & z/OS 1.11

V7 Performance Summary

Substantial improvements in performance of administration functions

For both small and large configurations

Connect and deploy

Cost of auditing and monitoring functions cut by 40%

Reduced virtual memory requirements at startup

Much easier and faster detection of performance problems

VSCR for z/OS users

Contents

Summary

- Design and coding are key
  - Tuning can help but is not a cure all
- Avoid the common problems
- Stay Current
  - Best practices
  - Software

Summary

N O T E S

- As we have seen in the course of this presentation there are many factors to consider in order to build a Message Broker implementation which is efficient and high performing.
- It is important to know what is required from the start. Find out what is important and what is not. Persistent messages might be required but message validation is not for example. Understand the essentials of processing right at the beginning of design. In several cases designers we have seen users opt to use the MRM XML parser when non of the additional features that that parser gives were needed. In stead it was possible to use the XMLNS or XMLNSC parsers which are noticeably quicker. By switching parser we were able to save significant parsing costs.
- Good message flow design is important. Focus on techniques to reduce parsing costs. Minimise tree copying. Look at message flow structure. Follow the guidance in the presentation. It lists the key areas to think about.
- Tuning is an important of overall performance. There are multiple aspects to consider:
  - Message flow coding and optimisations
  - Message Broker queue manager
  - Message Broker
  - The choice of machine type – speed and number of CPUs for example

  Applying tuning can make a significant difference to message throughput. However good tuning cannot get you out of a bad design. Get the design wrong and you will struggle to achieve your goals.
- Read and digest the list of common problems. Make sure they do not happen to you! Those have all happened for real.
- Message Broker V6 and V6.1 provide significant performance boosts compared with earlier versions of the product. I would strongly recommend that you migrate to it if you have not already done so. You can get most of the performance gains by “simply” migrating to the new release. No application changes are required to any of your flows. Performance improvements in areas like ESQL, parsing, aggregation will be picked up automatically once you start running with the V6 runtime. There are further improvements to be obtained if you want to want to re-write some of message flows. The extensions to the CAST clause allow much more efficient date/time processing to be completed for example. Using the MQGET node within a message flow can mean that it is possible to store state data in an MQ queue, whereas previously you would have needed to use a database. This can give significant performance improvements.

- By using a combination of the material in this presentation and the latest version of the product you have the knowledge and function to develop high performing message flows. Good Luck.

Additional Information

- WebSphere Message Broker Performance Supportpacs:

All Performance Reports are available at
http://www.ibm.com/support/docview.wss?rs=171&uid=swg27007150&loc=en_US&cs=utf-8&lang=en

- developerWorks - WebSphere Message Broker Home Page
  http://www.ibm.com/developerworks/websphere/zones/businessintegration/wmb.html

- developerWorks Articles: (see notes for URLS)
  - What's New in WebSphere Message Broker V6.1
  - Determining How Many Copies of a Message Flow Should Run Within WBI Message Broker V5
  - How to Estimate Message Throughput For an IBM WebSphere MQ Integrator V2.1 Message Flow
  - Optimising DB2 UDB for use with WebSphere Business Integration Message Broker V5
  - How to monitor system and WebSphere Business Integration Message Broker V5 resource use
  - The Value of WebSphere Message Broker V6 on z/OS
  - JMSTransport Nodes in WebSphere Message Broker V6
  - Connecting the JMS Transport Nodes for WebSphere Message Broker v6 to Popular JMS Providers
  - HTTP transport nodes in WebSphere Message Broker V6
  - Testing and troubleshooting message flows with WebSphere Message Broker Test Client

Notes

- The V6.1 performance reports are available on the IBM internet site for free. The report numbers and URL are given on the foil.
- The WebSphere Business Integration Zone on developerWorks has a lot of material on Message Broker and business integration software in general and is well worth a visit.
- A number of articles have been written on various aspects of Message Broker. These are written to help you understand how to use certain features as well as tune and configure Message Broker. The articles and their URLs are as follows:
  - What's New in WebSphere Message Broker V6.1 at

http://www.ibm.com/developerworks/websphere/library/techarticles/0802_dunn/0802_dunn.html

  - What's new in WebSphere Message Broker V6 at

http://www.ibm.com/developerworks/websphere/library/techarticles/0510_dunn/0510_dunn.html

  - Determining How Many Copies of a Message Flow Should Run Within WebSphere Business Integration Message Broker V5 at http://www.ibm.com/developerworks/websphere/library/techarticles/0311_dunn/dunn.html
  - Optimizing DB2 UDB for use with WebSphere Business Integration Message Broker V5 at http://www.ibm.com/developerworks//websphere/library/techarticles/0312_acourt/acourt.html
  - How to monitor system and WebSphere Business Integration Message Broker V5 resource use at http://www.ibm.com/developerworks/websphere/library/techarticles/0407_dunn/0407_dunn.html
  - The Value of WebSphere Message Broker V6 on z/OS at http://www.ibm.com/developerworks/websphere/library/techarticles/0604_odowd/0604_odowd.html
  - JMS Transport Nodes in WebSphere Message Broker V6 at

http://www.ibm.com/developerworks/websphere/library/techarticles/0604_bicheno/0604_bicheno.html

  - Configuring and tuning WebSphere MQ for performance on Windows and UNIX at

http://www.ibm.com/developerworks/websphere/library/techarticles/0712_dunn/0712_dunn.html

  - Connecting the JMS Transport Nodes for WebSphere Message Broker v6 to Popular JMS Providers at http://www.ibm.com/developerworks/websphere/library/techarticles/0610_bicheno/0610_bicheno.html
  - HTTP transport nodes in WebSphere Message Broker V6 at

http://www.ibm.com/developerworks/websphere/library/techarticles/0608_braithwaite/0608_braithwaite.html

  - Testing and troubleshooting message flows with WebSphere Message Broker Test Client at

http://www.ibm.com/developerworks/websphere/library/techarticles/0702_spriet/0702_spriet.html

Additional Information…

- Other SupportPacs:
  - IH03 - WBI Message Broker V6 - Message display, test and performance utilities
  - IP13 - WebSphere Business Integration Broker - Sniff test and Performance on z/OS

We Value Your Feedback !

Please complete the session survey for this session by:

Accessing the SmartSite on your smart phone or computer at: http://imp2010.confnav.com

– Surveys / My Session Evaluations

Visiting any onsite event kiosk

– Surveys / My Session Evaluations

Session TCE-1346A

Each completed survey increases your chance to win an Apple iPod Touch with daily drawing sponsored by Alliance Tech