Introduction to WebSphere Message Broker - your Universal ESB

David Coles – WebSphere Message Broker Level 3 Service,
IBM Hursley – dcoles@uk.ibm.com

Tuesday 3rd August 2010

- Welcome to this Technical Introduction to WebSphere Message Broker.
- All slides in this presentation have at least one corresponding notes slide like this one, which contains further information on the topic being discussed, and/or links to web pages.
- Only this notes slide will be shown during the presentation. To view all other notes slides, please download and view a copy of this presentation.
- The WebSphere Message Broker homepage can be found at http://www.ibm.com/software/integration/wbimessagebroker/

Agenda

- What is Message Broker?
- Programming Concepts
  - Message Flows
  - Nodes
  - Message Model
- Product Overview
  - Components
  - User Roles and Environments
  - Connectivity Scenarios
- Demonstration

- This presentation is divided into several sections. We'll begin by describing what Message Broker is and why it is important. At this time we'll also go through the steps required to integrate applications together.

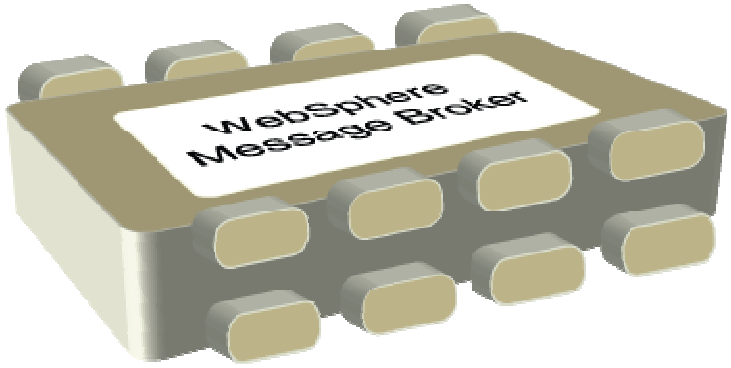
- Message Broker exposes three important concepts that allow you to integrate applications together. We see the objects that WebSphere Message Broker supplies to help integrators achieve a rapid solution. We'll introduce message flows and nodes which allow the connectivity and functionality requirements of applications to be modelled and built. We'll then explain the logical message model, a concept which allows us to manipulate messages without having to understand the physical details of the message format.

- We'll then cover a very brief overview of the components that make up WebSphere Message Broker, and the typical user roles and deployment environments associated with it. We'll also look at several typical scenarios that Message Broker is used for.

- Finally, if we have time a demonstration will help bring everything together.

What is WebSphere Message Broker?

- Message Broker enables “universal connectivity” by integrating protocols, message formats and mediation patterns
  - Emphasis on application re-use
- Fits naturally with WebSphere MQ
  - Robust, scalable architecture
  - Optimized for high throughput
  - Flexible broker topologies
- Three programming constructs are used:
  - Message Flows
  - Nodes
  - Message Models



- WebSphere Message Broker is IBM's strategic product for "Universal Connectivity". Its aim is to simplify application connectivity to provide a flexible and dynamic infrastructure, that routes and transforms messages FROM anywhere, TO anywhere.

- IBM has two other strategic ESBs that solve different connectivity problems: WebSphere ESB provides an integrated SOA platform built on WebSphere Application Server, and WebSphere DataPower XI50 is a purpose-built hardware ESB for simplified deployment and hardened security. More on this later.

- There are three parts to application integration: protocols, message formats and mediation patterns.

- Message Broker exposes three programming concepts in order to integrate applications: message flows, nodes and message models.

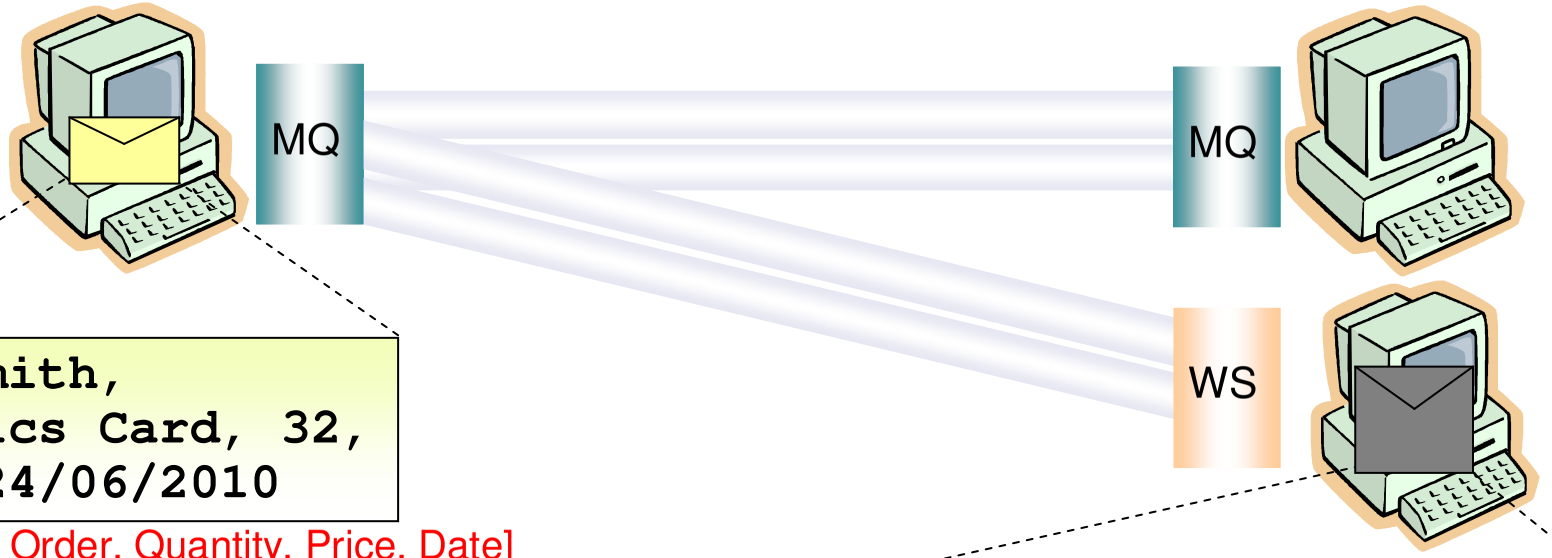
- References:
  - The Message Broker home page can be found at http://www.ibm.com/software/integration/wbimessagebroker/
  - The documentation for Message Broker can be found at http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp

Application Connectivity

- Protocols
  - e.g. MQ, TCP/IP, HTTP, File system, FTP, SMTP etc.
- Message Formats
  - e.g. Binary (C/COBOL), XML, Industry (SWIFT, EDI, HL7), User-defined
- Mediation Patterns
  - e.g. Route, Transform, Enrich, Filter, Monitor, Distribute, Decompose, Correlate, Fire and Forget, Request/Reply, Publish/Subscribe, Aggregation, Fan-in, Complex Event Processing

- Three strands are involved in connecting applications together.
- Applications need to talk with each other over a communications protocol. Typical protocols in use today include TCP/IP, and higher level protocols such as FTP, SMTP and HTTP.
- Over the communications protocol applications exchange data, typically in discrete structures known as messages. The format of these messages can be defined from C structures or COBOL copybooks (for example), or simply use a standard format such as XML.
- In order to connect applications together so that their protocols and message formats interoperate, mediation patterns need to be applied to one or both systems you're trying to connect. These mediation patterns can be relatively straightforward, e.g. routing messages from one place to another, or the transformation of one message format into another… to relatively complex patterns such as aggregating multiple outputs from an application into a single message for a target system.

Mediation Patterns – Routing and Transformation




```
Mr. Smith,
Graphics Card, 32,
100, 24/06/2010
```

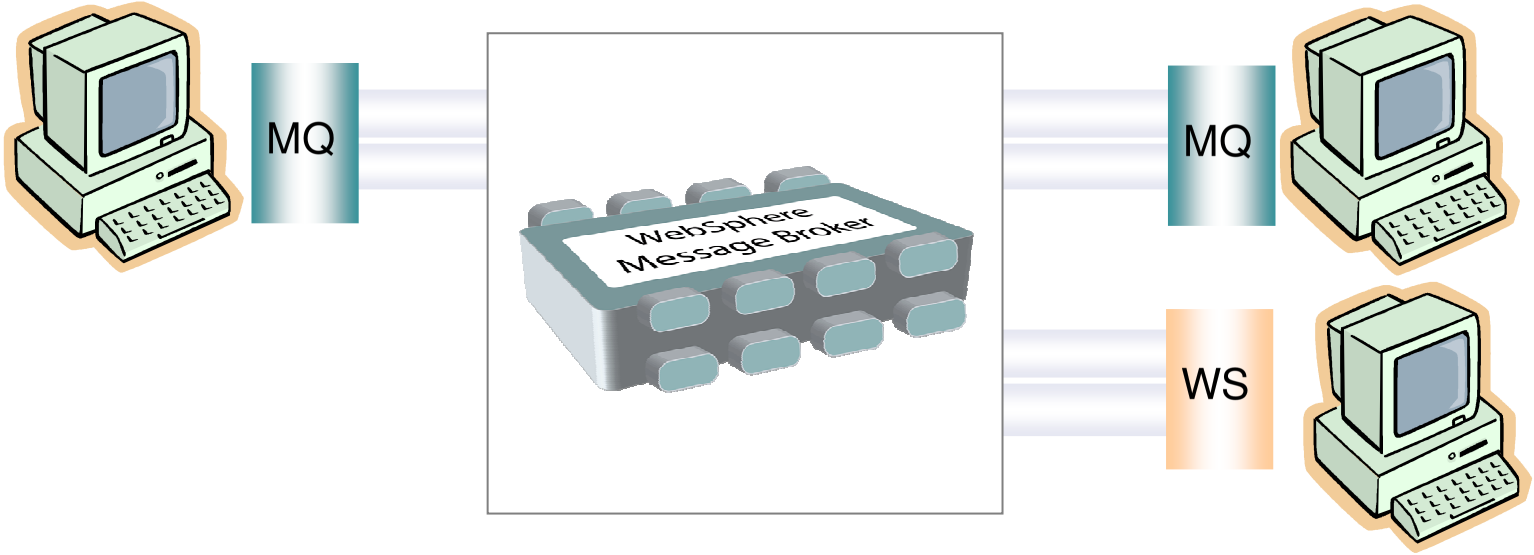
[Customer, Order, Quantity, Price, Date]

```
<order>
  <name>
    <first>John</first>
    <last>Smith</last>
  </name>
  <item>Graphics Card</item>
  <quantity>32</quantity>
  <price>200</price>
  <date>06/24/2010</date>
</order>
```

[Customer, Order, Quantity, Price, Date]

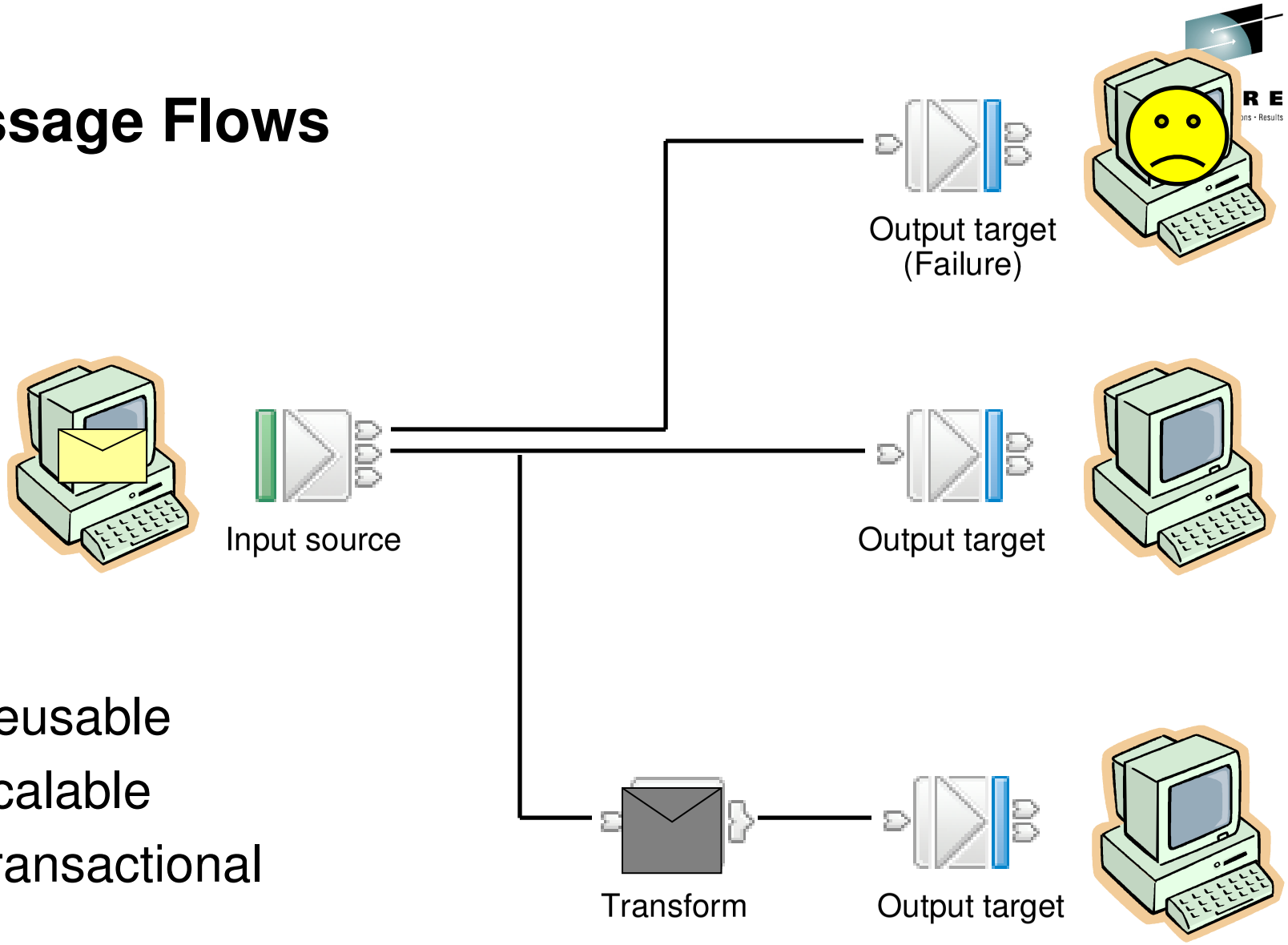
- An Application Integration Scenario.
- Application A sends some data to application B. At design time, the two applications agreed on the format of the data as the ordered set {Customer, Order, Quantity, Price, Date}. Further, the date is in UK format, the price in UK pounds sterling, and all fields are represented by character strings in codepage 500. Finally, the data is delimited using commas.
- Some time later, Application C is introduced. It needs the same data, but because it is a packaged application from a vendor or may be an application that already existed, it expects data to arrive in a different format. The date is in US format, the price is in dollars and the data is in XML.
- So, we now have an integration choice to make. Either application C must be enhanced to support the data format between A and B, or application A must be enhanced to support application C's data format. (This is an interesting use of the word “enhanced”, but you'll probably want to use it to justify the expenditure!)
- By introducing a solution that can mediate between these applications, you can integrate them without spending time and money modifying and retesting the existing applications. Message Broker is one such solution.

Application Connectivity with WMB



- WMB can act as an intermediary
- Flexible topologies

Message Flows



- Reusable
- Scalable
- Transactional

- Message flows provide the processing sequence required to connect applications together.

- A message flow contains the set of operations required to take a message from an originating application and deliver copies of it, some possibly transformed, to any number of connected applications for processing.
- As a message passes through a message flow, it is transformed and routed according to the nodes it encounters, and the processing decisions made within those nodes. Later we'll see how the nodes can modify the values within, or transform the structure of, a message to provide the data transformations necessary to drive backend server applications.
- For a given application scenario, the message flow describes all possible outcomes when processing a message. For example, if the message has a high monetary value, a copy of it might have to be routed to an audit application. Or if the message is not well-formed (may be it's not encrypted in the right format), it might be routed to a security application to raise an alert.
- Equally important is the visualization of the application integration within then organization. Very often, for any particular application scenario, the application connectivity requirements (*business*!) is held within the heads of domain experts. Being able to view the integration structure brings benefits in scenario understanding, reuse potential, and application architecture/standards conformance.
- After a message has been processed by a message flow, the flow does not maintain any state. It is possible to maintain such state in an external database, or within the message by using an extensible header such as the MQRFH2.

- Message flows are general purpose, reusable integration applications.

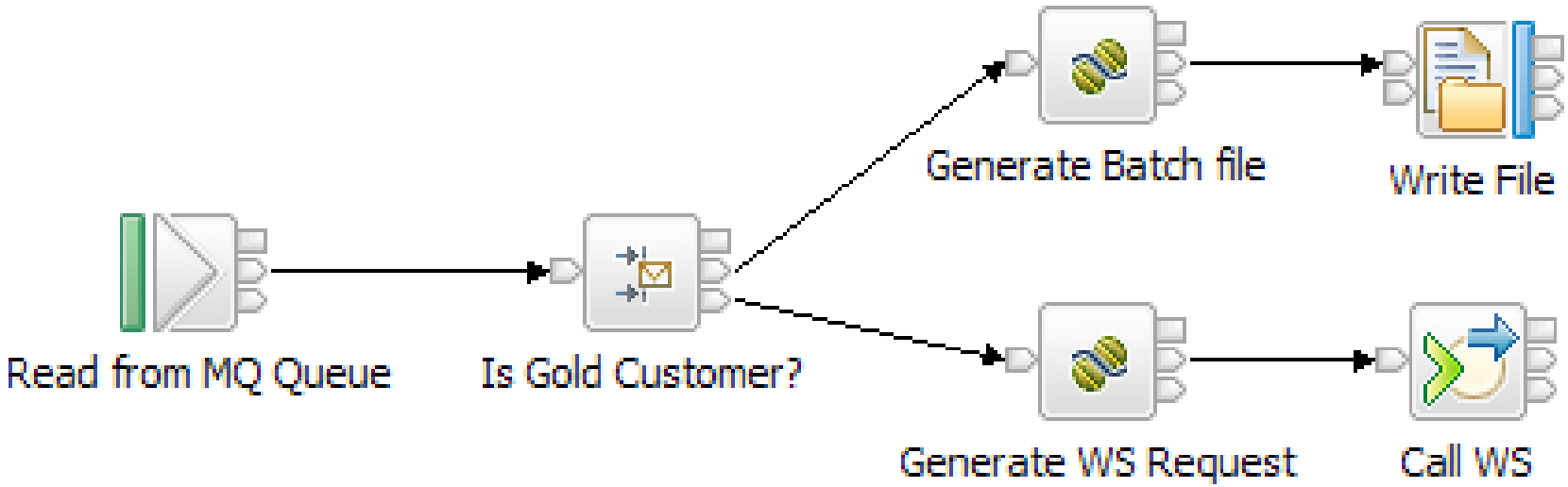
- If you were designing a general purpose integration application, linking client and server applications, the logic would comprise separate routines each performing a well-defined function. The input routine would wait for a message, and after receiving it and checking the its integrity, (well formed etc.), it would transfer to the next routines to continue processing.

- After performing their processing, (e.g. enriching/reformatting/routing), these routines would pass control on through to the lowest functional levels, where output processing would occur. Here, messages would be written to devices, subsequently read by connected applications. At any level of processing an exception could be raised for subsequent processing.

- After the last output routine had completed, control would return back up through the levels to the input routine. Once here, all the changes would be committed and the input routine would wait for more input.

- Message flows are transactional.
  - Message flows provide vital processing and data manipulation and are therefore fully transactional. A message flow either completes all or none of its processing successfully.
  - However, if required, individual nodes can elect to perform operations outside of the message flow transaction. (e.g. audit)
- Message flows are multithreaded.
  - A given message passing through a series of nodes will execute on a single thread. To allow increased message throughput, message flows can be defined with many additional threads assigned to them. Peak workloads use additional threads, which are pooled during inactivity. We'll see more implementation details later. This means application scaling can be an operational rather than design time decision.
- Message flow nesting and chaining allow construction of enhanced capabilities.
  - Sophisticated flows can be rapidly constructed by linking individual flows together as well as nesting flows within each other.
- References:
  - Message Flow overview at http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac00310_.htm

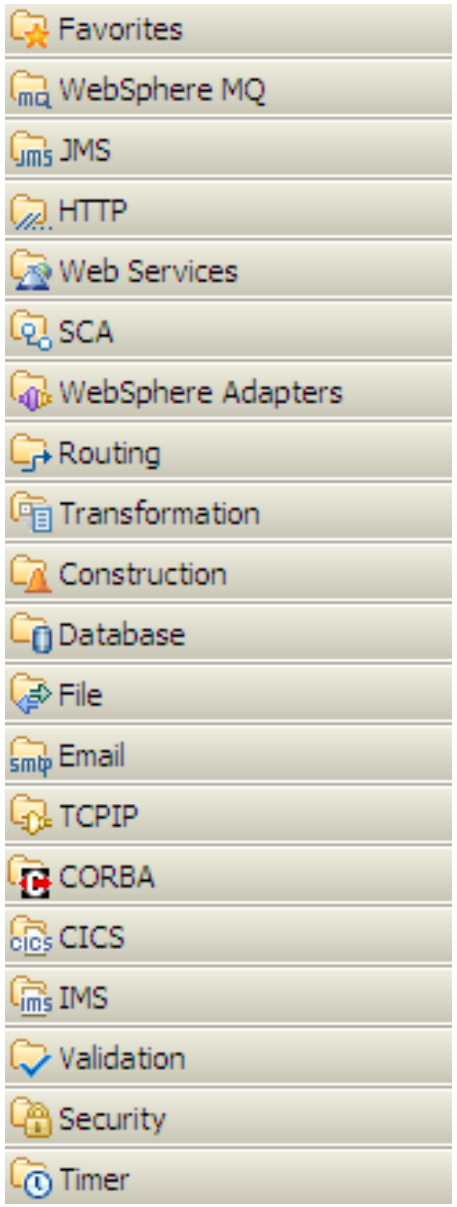
Message Flow Example



- Here is an example of a message flow.
- The In ‘Read from MQ Queue’ node tells the Message Broker to takes messages from an MQ queue (the name of which is embedded as a property of the node, or overridden by an administrator at deployment time).
- The message is passed onto the ‘Is Gold Customer?’, where a routing decision is made based on a field described in the incoming message, again which is a property on the node itself. We’ll see exactly how this condition is specified later on.
- If the described condition holds, the message is routed to the ‘Generate WS Request’ node where the message is transformed – presumably into an SOAP message that is recognisable by the web service which is invoked by the subsequent ‘Call WS’ node.
- If the described condition does not hold, the message is routed to the ‘Generate batch file’ node, which formats the message for subsequent output to a file in the ‘Write file’ node.
- This flow may not tell the complete integration story between the calling application and the target Web Service/File applications. For example, there is no communication back to the calling application to say that the message has been processed (or even received). Nor is there any logic in the message flow to cope with failures – for example, if the web service is not available. This is logic that could be incorporated into the message flow, but not visualised here for clarity.

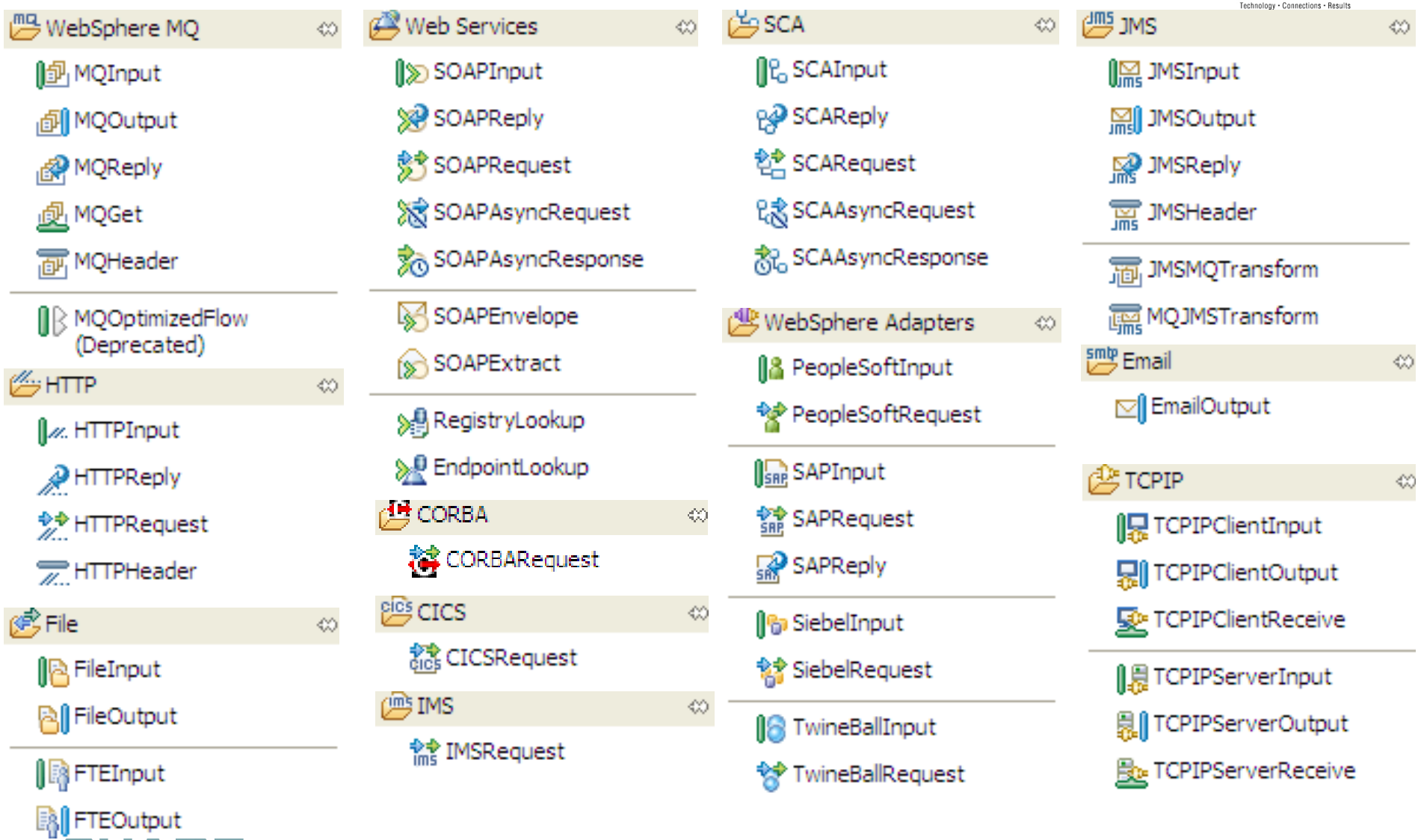
Nodes

- The building blocks of message flows
- Each node type performs a different (input, output or processing) action
- Many different node types
  - Grouped into logical categories in the message flow editor
  - Nearly 100 nodes available out-of-the-box
    (as of WMB V7)



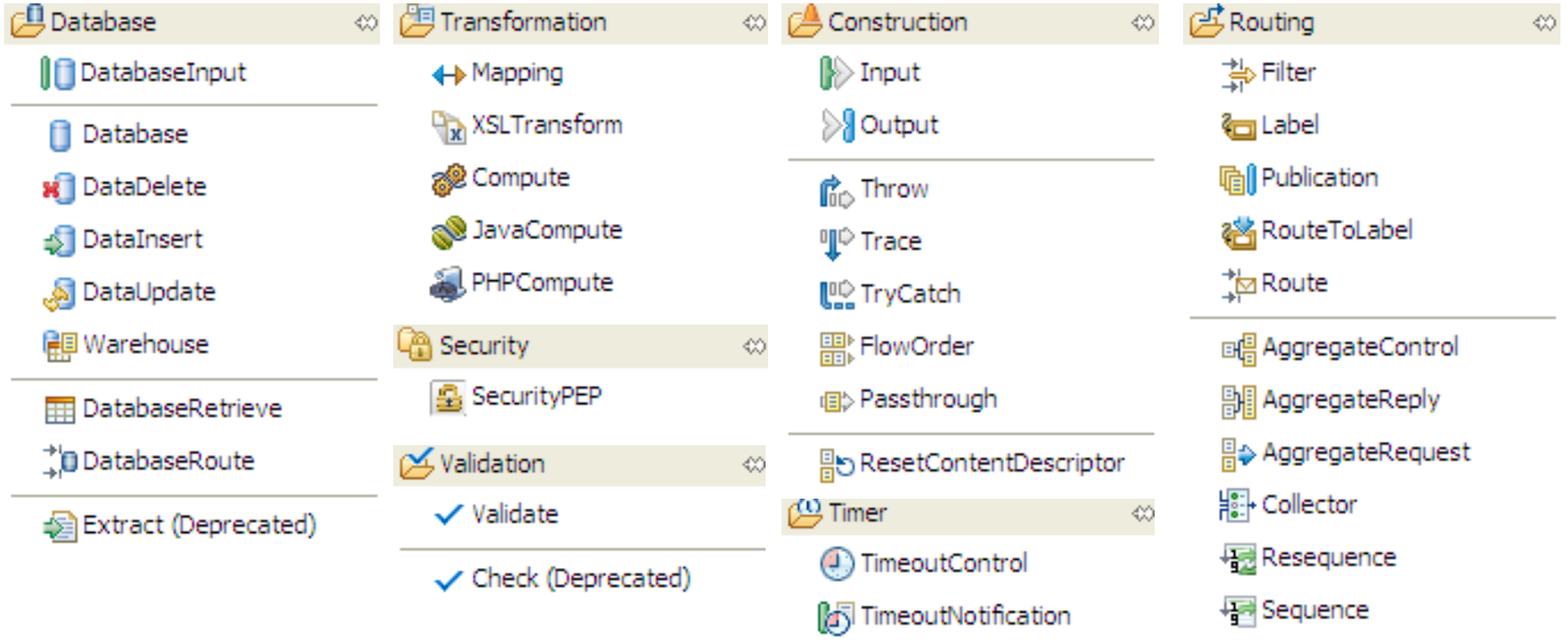
- Nodes can be grouped in several ways; for example, by where in the flow they are used:
  - Input nodes do not have input terminals; processing of the message flow starts when a message is retrieved from an input device, for example WebSphere MQ.
  - Output nodes do not have output terminals (or at least, they are not wired to any other node). The final stage of output processing is after a message is put using one or more output nodes, and processing control returns to the input node which commits or backs out the transaction. Recalling that a message flow is analogous to a functional decomposition, it makes sense that the top most level (i.e. the input node) controls the overall transaction.
  - Processing nodes are nodes that are neither input nor output nodes. They will be connected to nodes both upstream (i.e. towards the input nodes) and downstream (i.e. towards the output nodes).
- They can also be grouped by the function that they perform.
  - Protocol-specific nodes give the broker the ability to interact with particular systems, such as MQ and Web Services.
  - Transformation nodes will take a message in one format on the input terminal and output a converted message on the output terminal.
  - Logical constructs give the message flow designer the vocabulary required to solve complex integration scenarios, for example, the ability to aggregate messages from multiple places or the ability to filter messages based on their content.
- References
  - More on nodes can be found here: http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac04550_.htm

Lots of Nodes are Built in [1]



- Here's a list of the protocol specific nodes built in to Message Broker V7. For example:
  - The WebSphere MQ nodes allows Message Broker to interact with queues on MQ and MQe queue managers. For example, MQInput is an input node that triggers a flow when a message arrives on a queue; MQOutput puts a message to a queue.
  - The WebSphere Adapters nodes provides native support in Message Broker for inbound and outbound communication with Enterprise Information Systems.
  - Web Services nodes provide a rich environment for running as a Web Services requestor, provider and intermediary. Support for WS-Security, WS-Addressing, import and export of WSDL and validation against the WS-I Basic profile. The RegistryLookup and EndpointLookup nodes provide support for WebSphere Registry and Repository (WSRR).
  - HTTP nodes complement the Web Services capability. Support is provided for HTTP 1.0, 1.1 and HTTPS.
  - JMS nodes work with *any* JMS 1.1 compliant provider.
  - The EmailOutput node is a highly configurable node that allows e-mail messages to be sent over the SMTP protocol.
  - TCP/IP nodes allow the Message Broker to communicate with any client or server talking the ubiquitous TCP/IP protocol.
  - CORBA, IMS and CICS request nodes for integrating with CORBA, IMS and CICS applications respectively.

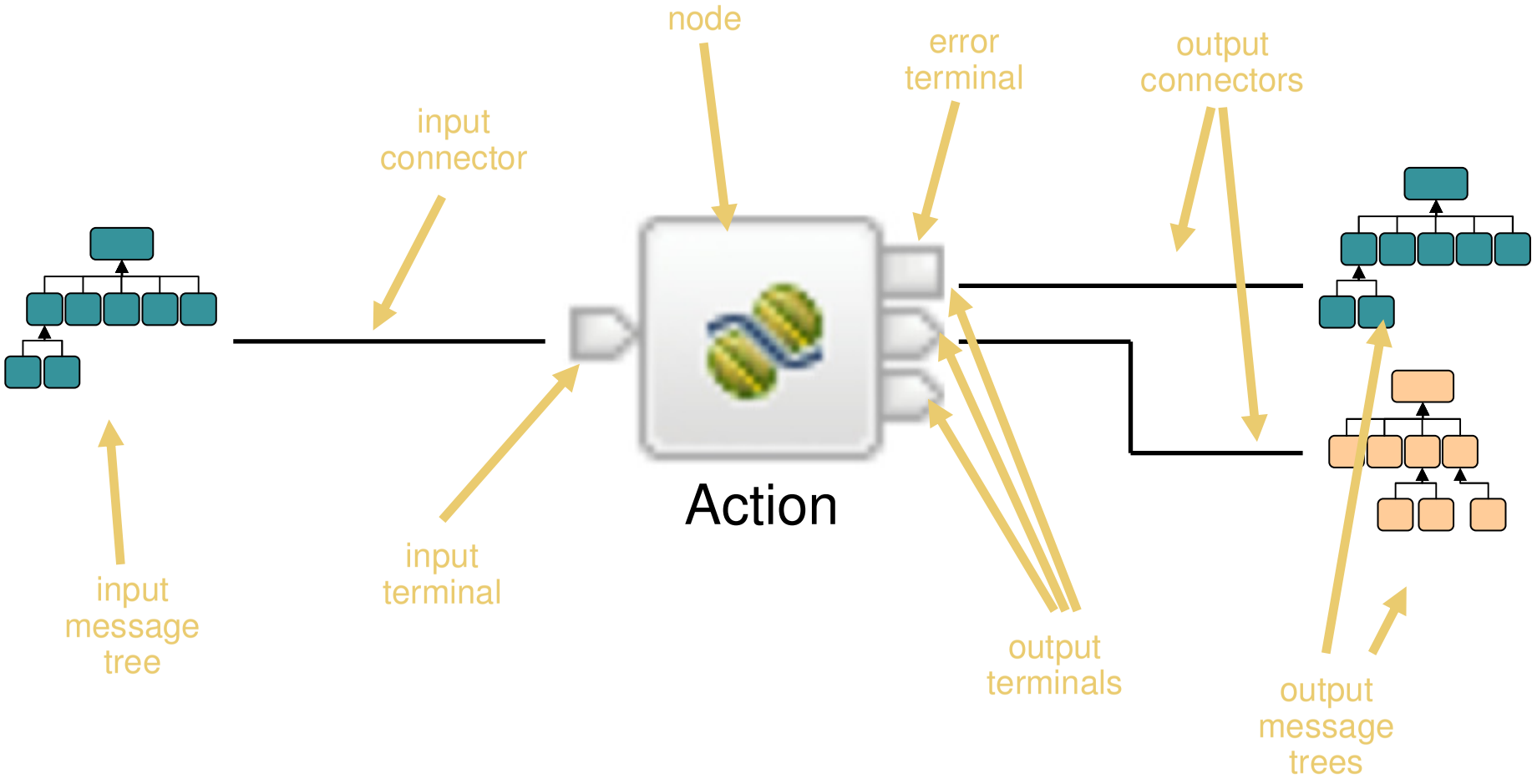
Lots of Nodes are Built in [2]



- Many other nodes available through product extensions and supportpacs
  - For example, WebSphere TX, Tibco RV, VSAM, QSAM
- Write your own User-Defined Nodes in C or Java

- Continuing the discussion on the nodes inside WebSphere Message Broker:
  - Database nodes allows message flows to interact with many different data sources, including DB2, Oracle and Sybase.
  - Timer nodes provide support for triggering message flows and certain times or intervals.
  - The Routing category allows messages to easily flow around a network, and also allow multiple messages to be aggregated or propagated in the correct sequence.
  - File nodes allow messages to be read from, or written to the local file system or an FTP server. MQ File Transfer Edition nodes embed an FTE Agent inside the broker to allow it to perform processing on the back of a file transfer action.
- The Transformation category provides Message Broker with the capability to transform messages from one format into another. Five ways of doing this are available out-of-the-box. More on these later.
- Construction nodes
  - Nodes have error handling as part of their design. If an error is detected within a primitive node (e.g. database error), the message is transferred to the failure output terminal. If the failure terminal is not connected, an exception is generated and propagated back towards the input node. There is also a specialized Throw node which allows a flow designer to generate an exception in a controlled way. Nodes can have transaction scope inside or outside of the flow.
  - A TryCatch node is used to process any such exceptions. Its ‘try’ terminal is used for normal processing, but if an exception occurs along this path, the TryCatch node regains control and the original message is propagated through the ‘catch’ terminal.
  - If the message reaches the input node, it is subject to "back out" processing. In this case, it will be propagated down its catch or failure terminal, returned to the input queue, put to a back out or dead letter queue, or discarded, as appropriate.

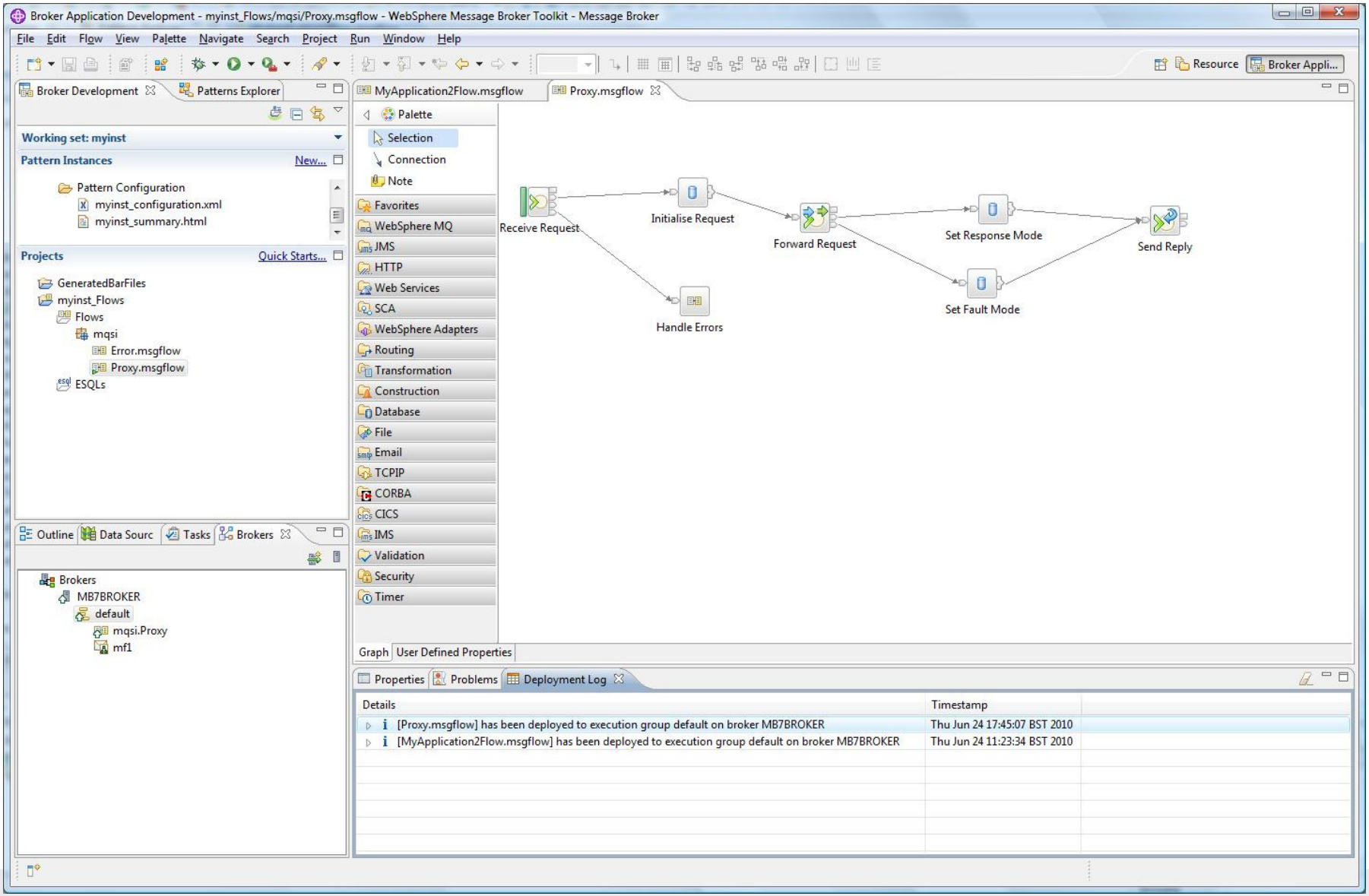
Node Terminology



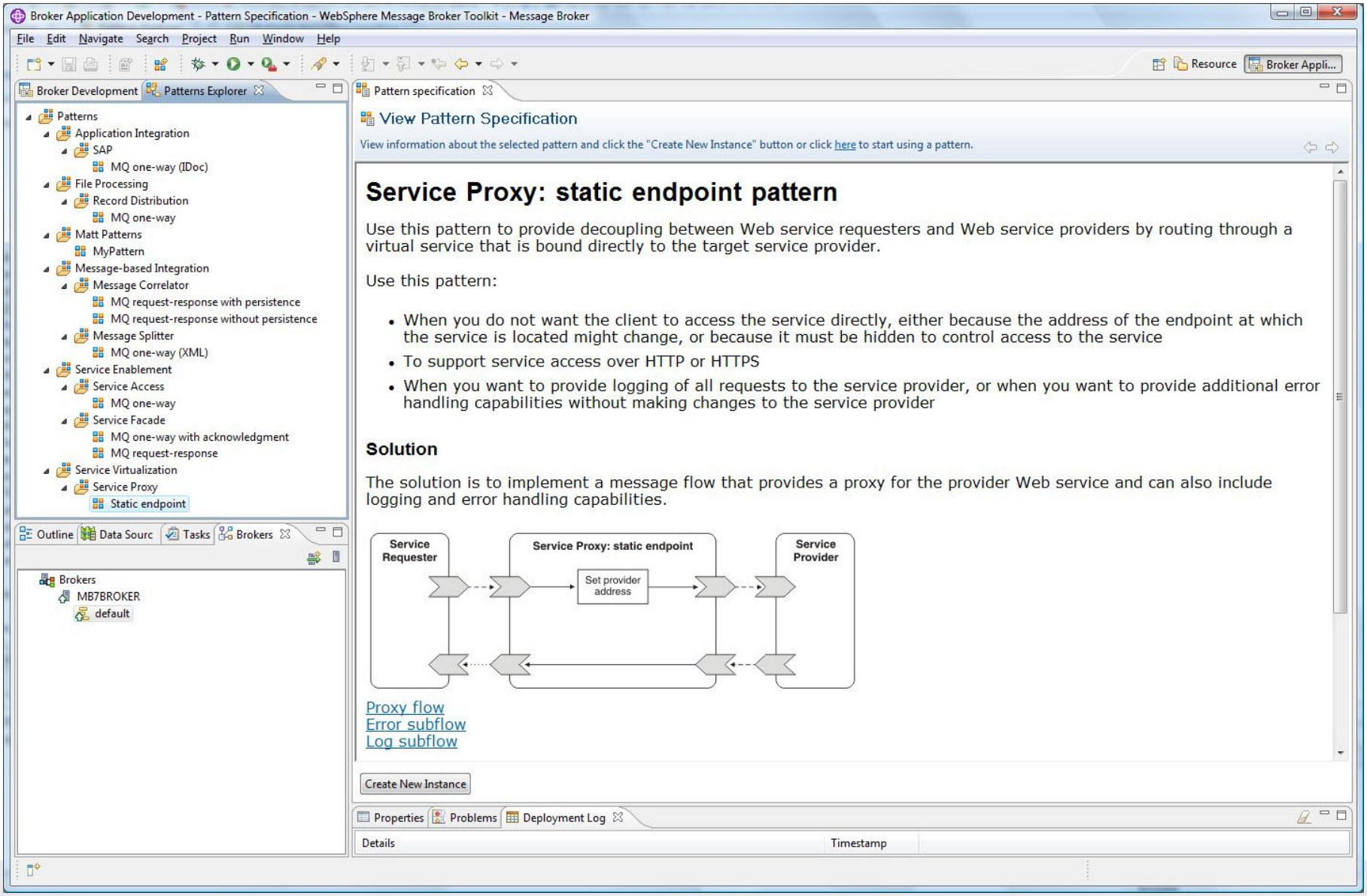
- Message nodes provide the individual processing elements that make up a message flow.
- We've seen that a message flow is the combination of operations required to achieve application integration. We build a message flow from small units called nodes; these nodes represent the base elements required to connect messaging applications together.
- Looking at a message flow, you can see several objects identifiable with this processing.
  - Nodes represent functional routines encapsulating integration logic
  - Terminals represent the various outcomes possible from node processing
  - Connectors join the various nodes through their terminals
- A message processing node defines a single logical operation on a message.
- A message processing node is a stand alone procedure that receives a message, performs a specific action against it, and outputs zero or more messages as a result of the action it has taken.
- The action represented by a message processing node encapsulates a useful and reusable piece of integration logic. Nodes can be thought of as reusable components in an integration library.

- A node is joined to its neighbours in the data flow through connectors attached to its data terminals.
- Every node has a fixed number of connection points known as "input" terminals and "output" terminals. These allow it to be connected to its neighbours. Each node normally has one input terminal upon which it receives messages, and multiple output terminals for different processing results within the node. Different types of node have different numbers of terminals.
- A connector joins an output terminal of one node to an input terminal of the next node in the message flow. You can leave an output terminal unconnected, or you can connect a single output terminal to more than one target node.
- After a node has finished processing a message, the connectors defined from the node's output terminals determine which nodes process the message next. If a node has more than one output terminal connected to a target node, it is the node (not you) that determines the order in which the different execution paths are executed. If a single output terminal has more than one connector to a target node, it is the broker (again, not you) which determines this execution order.
- A node does not always produce an output message for every output terminal: often it produces one output for a specific terminal depending on the message received. E.g. a filter node will typically send a message on either the true or false terminal, but not both.
- When the processing determined by one connector has been completed, the node reissues the message to the next connector, until all possible paths are completed. Updates to a message are never propagated to previously executed nodes, only to following nodes.
- The message flow can only start processing the next message when all paths through the message flow (that is, all connected nodes from all output terminals, as appropriate) have been completed.

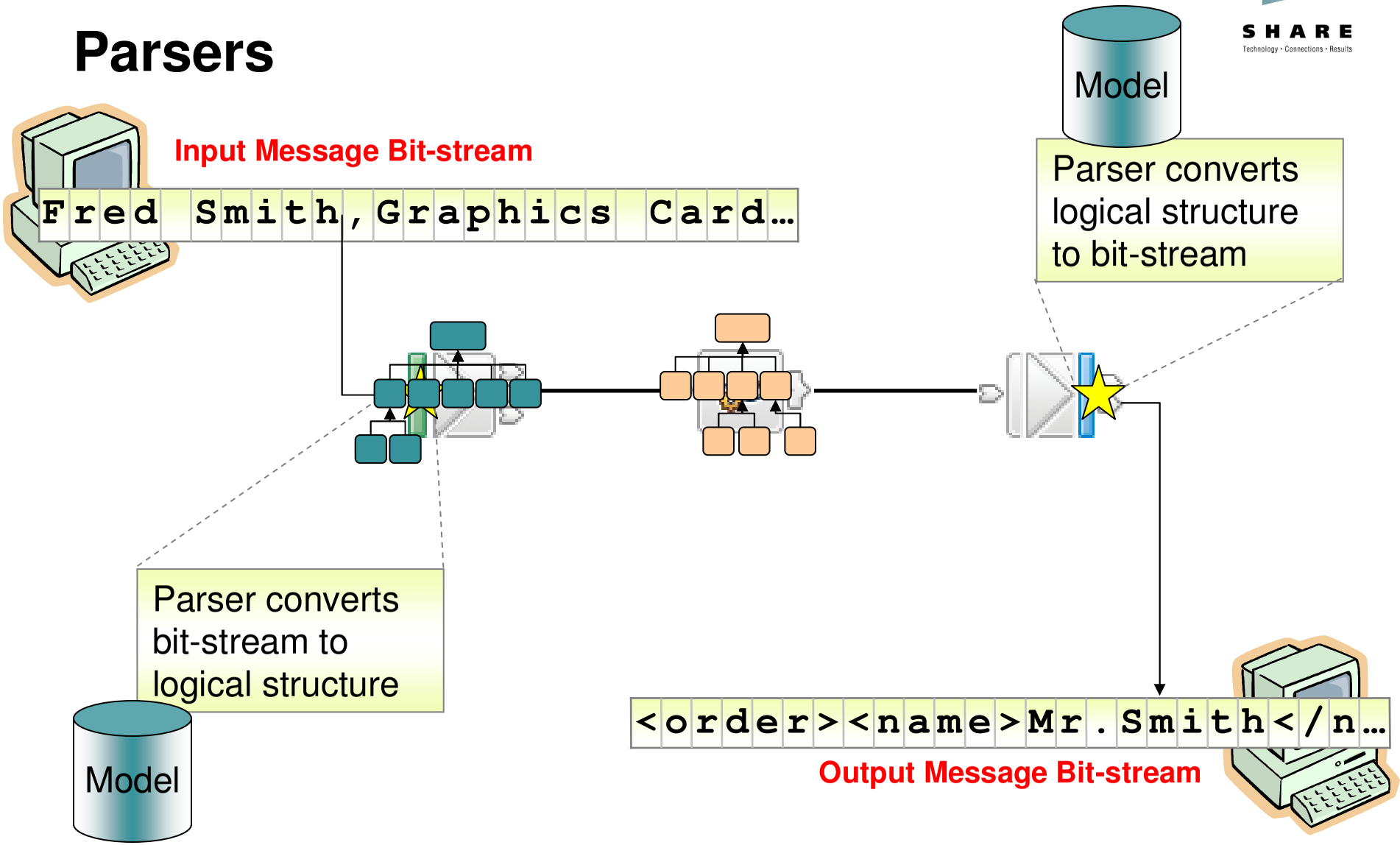
Message Flow Editor



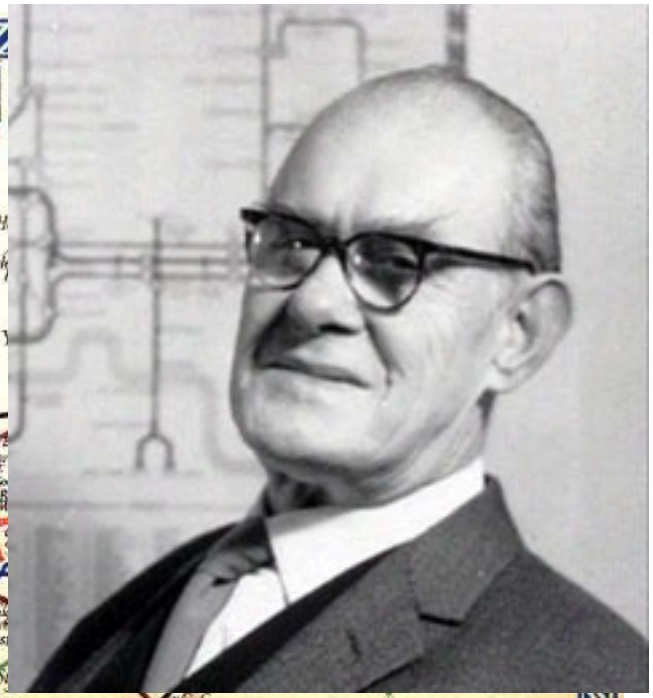
Message Broker Patterns



Parsers


Input Message Bit-stream
Fred Smith,Graphics Card…
Model
Parser converts logical structure to bit-stream
Parser converts bit-stream to logical structure
Model
<order><name>Mr.Smith</n…
Output Message Bit-stream

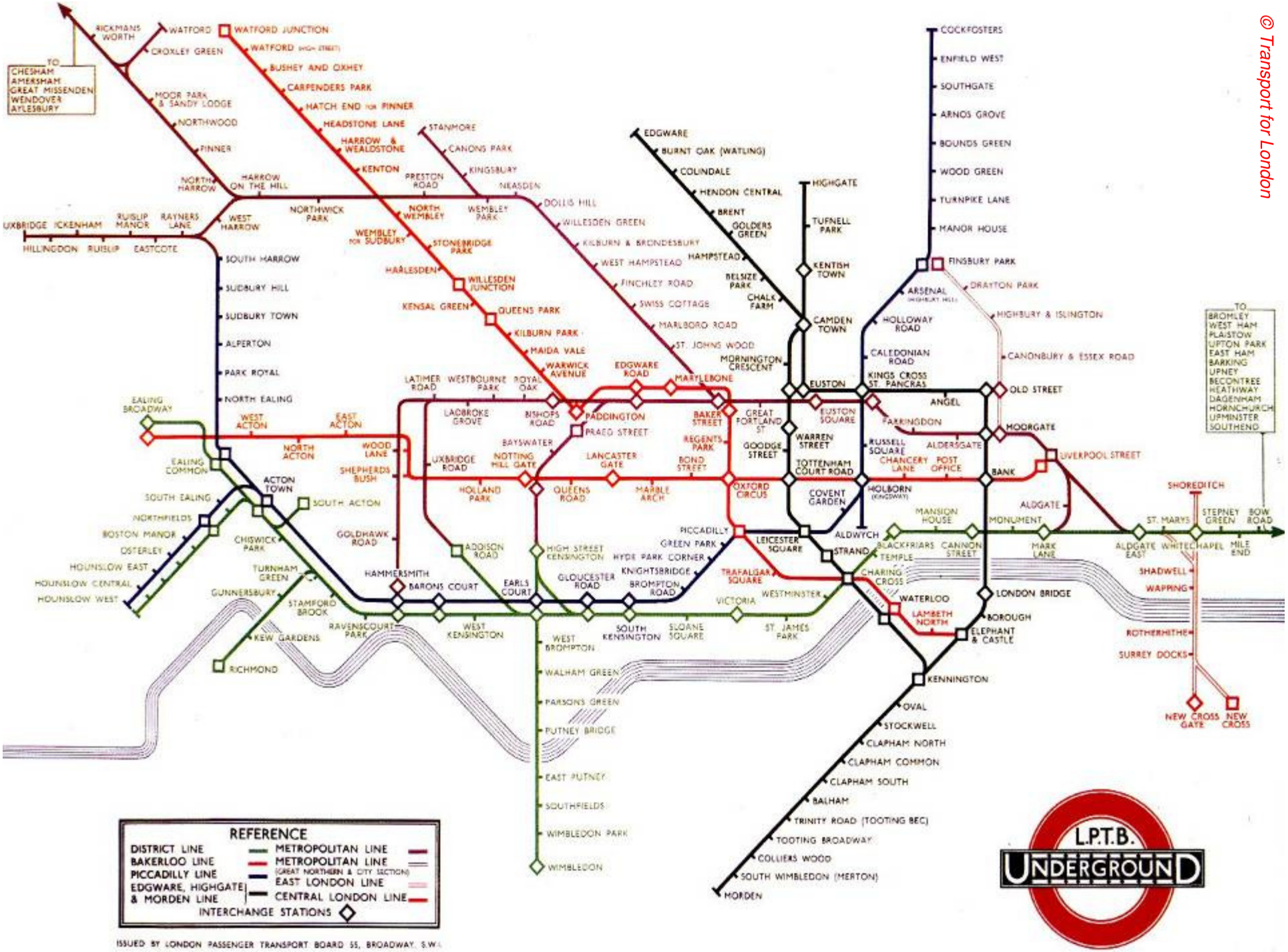
- On the previous slide we saw that objects called “message trees” are sent to a node’s input terminals, and either the same or different message tree is propagated from a node’s output terminals.
- The message tree is a logical definition of a message processed by the broker. It’s described as a tree because messages are typically hierarchical in structure; a good example of this is XML. Other message formats too, are also often derived from complex structures which themselves can be derived from complex structures, and so on, which gives them a tree-like shape with leaf nodes representing simple data types.
- In WebSphere Message Broker, parsers have the job of converting between physical messages (bit-streams) and logical trees. When a message arrives at the broker through an input node, the message bit-stream is converted into a tree structure by the parser, which typically uses a model to drive the form of the logical tree. Built-in parsers handle well known headers within the message (MQMD, MQRFH2 etc.). Finally the user data is parsed into the tree using the domain parser as identified in the MQRFH2 (or input node). Message Broker’s built-in parsers support multiple domains (MRM, SOAP, XMLNSC, Data Object, XMLNS, JMSMap, JMSStream, MIME, IDOC, BLOB and XML) to enable parsing of user and industry standard formats.
- As the logical tree is passed from node to node, the form of the logical tree may change depending on what the node is doing.
- When the message arrives at an output node, the parser converts the logical tree back into a physical bit-stream where it can be output to the external resource, where it can be read by the target (receiving) application. However, note that an output node need not indicate the end of a flow; it is possible to output to multiple destinations within a single invocation of a message flow. In this case, the logical tree can be passed on to other nodes and manipulated further, even after it has been converted back into a physical bit-stream for this particular output node.



- This is a map of the London Underground railway system from 1921. Notice how the lines relate to the physical layout of the stations rather than the logical layout. It makes it really difficult to work with this map, particularly around the central area.

- Harry Beck (pictured) produced a new style map in 1933. Beck was an Underground employee who realised that, because the railway ran mostly underground, the physical locations of the stations were irrelevant to the traveller wanting to know how to get to one station from another — only the topology of the railway mattered.

- See http://en.wikipedia.org/wiki/Harry_Beck_%28graphic_designer%29 for more information.



- This is Harry's map from 1933.
- He had devised a vastly simplified map, consisting of stations, straight line segments connecting them, and the Thames; lines ran only vertically, horizontally, or at 45 degrees. The Underground was initially sceptical of his proposal — it was an uncommissioned spare-time project, and it tentatively introduced it to the public in a small pamphlet. It was immediately popular, and ever since the Underground has used topological maps to illustrate the network.
- In Message Broker terms, Harry was the "parser" – he took the physical representation of the London Underground and converted into a logical structure. This logical structure is very much easier to work with.
- Message Broker uses logical structures to describe physical data for similar reasons: it makes them much easier to work with, particularly when addressing or converting between data elements. We'll see examples of this later on.
- See http://en.wikipedia.org/wiki/Tube_map for more information.

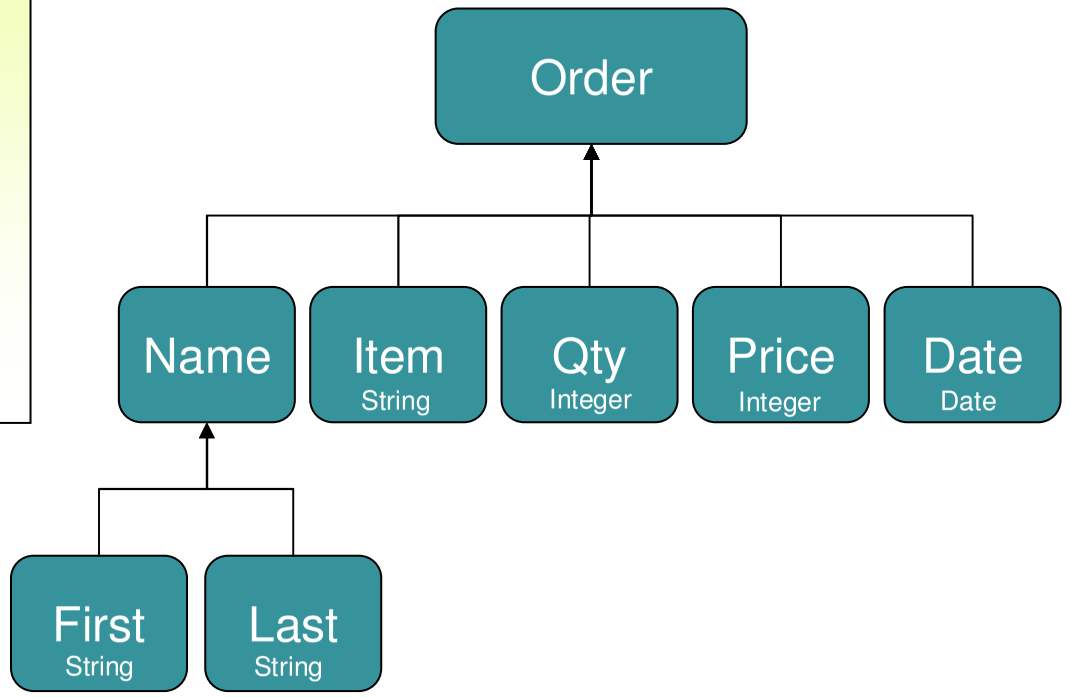
Message Modeling

Physical ⟷ Logical

```
<order>
  <name>
    <first>John</first>
    <last>Smith</last>
  </name>
  <item>Graphics Card</item>
  <quantity>32</quantity>
  <price>200</price>
  <date>07/11/09</date>
</order>
```

```
John,Smith,Graphics Card,
32,200,07/11/09
```

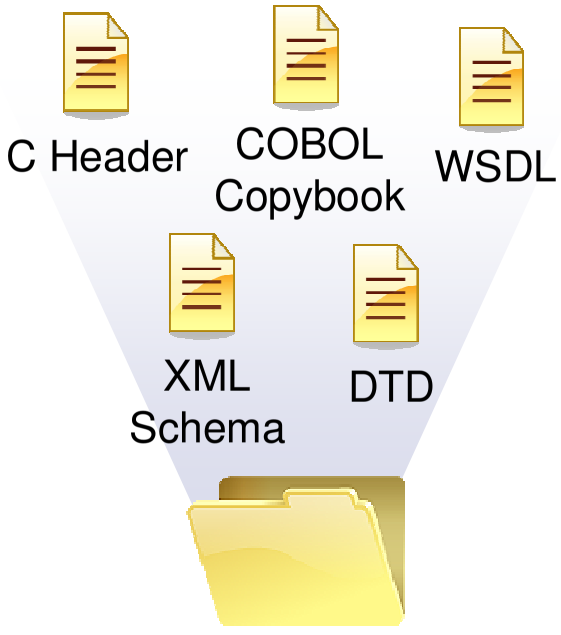
```
John Smith.............
Graphics Card..........
322002071109..........
```



- Here is an example of how a physical data structure could be mapped to a logical tree.
  - Notice how multiple physical formats can correspond to the same logical tree. The first physical format is an XML structure that shows our Order message. The second is a comma separated value (CSV) structure of the same. The third comprises a set of fixed length fields in a custom wire format.
  - By manipulating the logical tree inside the Message Broker rather than the physical bit-stream, the nodes can be completely unaware of the physical format of the data being manipulated. It also makes it easy to introduce new message formats into the broker.
- Applications have and require diverse data formats.
  - We all know that XML is the data format that's going to solve every data processing problem that exists! We also know that "XML++", the follow-on compatible meta format that someone in a research laboratory is working on will solve all the problems we don't even know we have today! The fact is that, without wanting to appear cynical, every generation goes through this process. Surely it was the same when COBOL superseded assembler.
  - The fact is, that for historic, technical, whimsical, political, geographical, industrial and a whole host of other reasons you probably never even thought of, a hugely diverse range of data formats exist and are used successfully by a myriad of applications every second of every day. It's something that we have to live with and embrace because it isn't going to get any better any time soon.
  - The advantage WebSphere Message Broker brings by modelling all these messages is that we can rise above the message format detail; so that whether it's a tag delimited SWIFT or EDIFACT message, a custom record format closely mapping a C or COBOL data structure, or good old XML, we can talk about messages in a consistent, format independent way. Message Broker can manage this diversity.

- The Logical Message Model.
  - Reconsider messages and their structure. When we architect messages (no matter what the underlying transport technology), we concern ourselves firstly with the logical structure. For example, a funds transfer message might contain an amount in a particular currency, a transaction date and the relevant account details of the parties involved. These are the important business elements of the message; when discussing the message, we refer to these elements.
  - However, when we come to realize the message, we have to choose a specific data format. This may be driven by many factors, but we have to choose one. You may be aware of the advantages of various message formats or have your own personal favourite, or may fancy inventing a new one, but the fact remains that you have to choose a physical *wire format*. So for our transfer message, we might decide to use XML, with its elements, attributes and PCDATA (and a DTD, if we're being really exact), or we might map more closely to a C data structure modelling our message with ints, shorts, chars etc. and worry about *their* various representations(!)
  - The Logical message model provided by WebSphere Message Broker allows one to describe a message in terms of a tree of elements, each of which has a (possibly user defined) type. At the message tree leaf nodes, the elements have simple types such as strings, integers, decimals, booleans etc. Moreover, elements can have various constraints and qualifiers applied to them that more fully describe them; e.g. elements might be optional, appear in a certain order or only contain certain values.

Creating Message Models



Enterprise Information System

(SAP, Siebel, PeopleSoft)

Pre-built

SOAP, MIME, CSV, IDOC, SWIFT, EDIFACT, X12, FIX, HL7, etc

Define your own

using the Eclipse-based Tooling

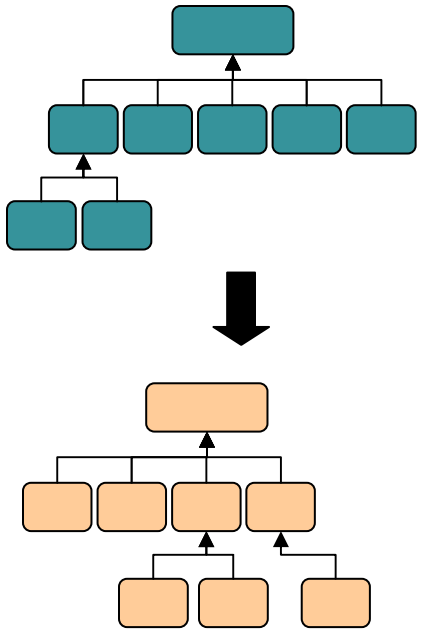
Message Set

Parsers

Message Broker

- This slide describes some of the options available for creating message models.
- Firstly, if you have messages described by COBOL copybooks, C header files XML DTDs/Schemas or WSDL, use the WebSphere Message Broker supplied importers to generate your message model. A wide range of importers exist, so that you can kick start your message modelling.
- Secondly, if you wish to use the SAP, Siebel or PeopleSoft nodes inside WebSphere Message Broker V6.1, you can construct message models directly from the Business Objects on these systems.
- Thirdly, you can use pre-built models such as those for SWIFT messages.
- Finally, you can use graphical modelling available in the Message Broker Toolkit to model your messages. You've seen application connections and processing constructed using message flows and nodes; the Message Broker Toolkit provides a similarly visual approach to message modelling.

Powerful Message Transformation Options



Compute

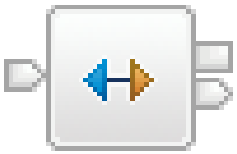
- Describe powerful transformations quickly
- Uses SQL-based language (ESQL)

JavaCompute

- Uses Java programming language
- Ability to use XPath

PHPCompute

- Transform using PHP scripts
- PHP 5.2 compliant

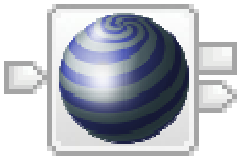


Mapping

- Graphical, easy to use
- Drag and Drop fields, apply functions

XSL Transform

- Convert XML to anything
- Uses standard XSL Style sheets



WTX Map

- Run a WebSphere Transformation Extender map

- There are several options available to you out-of-the-box for transforming between message formats:

- Use Graphical Mapping to visually represent messages and transform them. The Message broker has a mapping node to allows users to visualize and transform messages. The mapping node presents input and output message views; i.e. visualisation of message definitions. Users can “map” elements from the input message to the output message using “drag and drop” operations. More complex operations are possible, such as field concatenation. Graphical mapping is most effective when you have relatively simple transformations to perform and you don’t want to use a programming language (ESQL or Java).

- XSLT (eXtensible Stylesheet Language Transformations) is a language for describing message transformations. There is a node in WebSphere Message Broker that allows you to convert XML messages using style sheets developed in this language.

- The Compute node uses ESQL (Extended Structured Query Language). This is a language derived from SQL3, and is particularly suited to manipulating both database and message data. You do this with a single syntax (words) which has a common semantic (meaning). ESQL addresses message fields using a natural syntax. For example, the fourth traveller in a travel request message could be:

  InputRoot.Body.TravelRequestMessage.TravellerDetails[4].LastName

- ESQL has a rich set of basic data types and operators, as well as the kind of statements and functions you're used to from procedural programming languages, to allow you to perform powerful transformations.

- Continuing the options available to you for transforming between message formats:
- You can also use the power of Java to route and transform your messages. The JavaCompute node is fully integrated into the Eclipse Development Environment to providing usability aids such as content assist and incremental compilation. You can refer to elements using an expressive XPATH syntax, so that message navigation and element creation and modification are vastly simplified, and comparable in simplicity to ESQL field references. JDBC type 4 support allows you to perform database and message tree operations in the Java Compute node. On z/OS Java workload is eligible for offload onto the zSeries Application Assist Processors zAAP.
- You can also use PHP to transform your messages.
- As a separate product, you can also use WebSphere Transformation Extender maps to describe transformations.
- You can mix and match your transformation styles, or use just one throughout your enterprise. It will probably be your skill set which determines whether you chose to use Graphical Mapping, Java, XSLT, ESQL or PHP to perform your message routing and transformation.

Easily Address Message Elements


```
public class jcn extends MbJavaComputeNode {
  public void evaluate(MbMessageAssembly assembly) throws MbException {
    ...
    String lastName =
       (String)assembly.getMessage().evaluateXPath("/Body/Order/Name/Last");
    ...
  }
}
```

Route Node Properties - Route

Filter table*

Filter pattern	Routing output terminal
$Body/Order/Price > 1000	BigSpenders
$Body/Order/Price < 200	Cheapskates

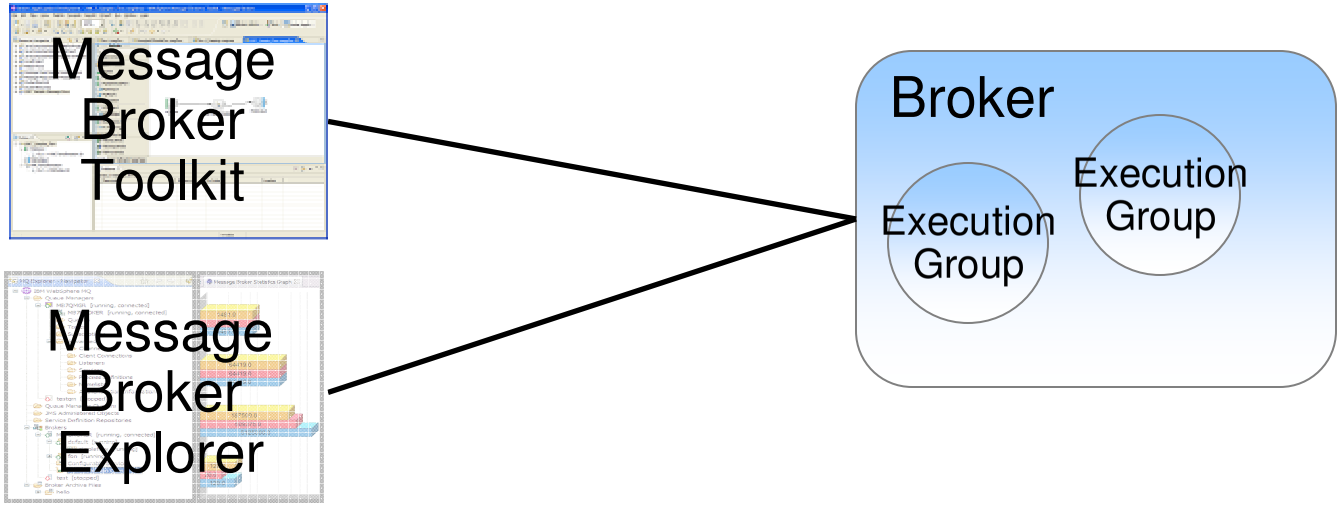

```
IF Body.Order.Date < '2008/01/01' THEN
   INSERT INTO Database.OldOrders (LastName,Item,Quantity)
   VALUES (Body.Order.Name.Last,
           Body.Order.Item,
           Body.Order.Quantity);
ENDIF;
```

- Each node's configuration (which includes some Java logic in the case of the Java Compute node) dictates what you want the node to do, and this may include manipulation of one or more elements in the message tree.

- Here are some examples of node configurations that address elements in the logical tree.

- In most cases, elements can be addressed using either XPATH (as shown in the JavaCompute and Route) or ESQL (as shown in the DataInsert node).

Architected for High Performance and Scalability



- Message Broker Toolkit
  - Development and Test Environment
  - Built on Rational Application Developer
- Message Broker Explorer
  - Advanced Administration Tool
  - Built on MQ Explorer
- Broker
  - Standalone runtime environment that runs message flows
  - Execution groups for isolation and scalability
  - Many different platforms
  - Builds on an MQ queue manager

- Now that we've understood the architectural components of WebSphere Broker, (message flows, nodes and the logical message model), we'll see how these elements are used within the brokers. Several different components form an operational broker.

- The three major components are the Message Broker Toolkit and the broker runtime, and the MB Explorer.

- WebSphere Message Broker Toolkit

- The WebSphere Message Broker Toolkit uses the Eclipse Platform, which is an extensible platform for developing Integrated Development Environments (IDEs). All the objects required to perform application integration using WebSphere Message Broker are developed, deployed and monitored inside this platform.

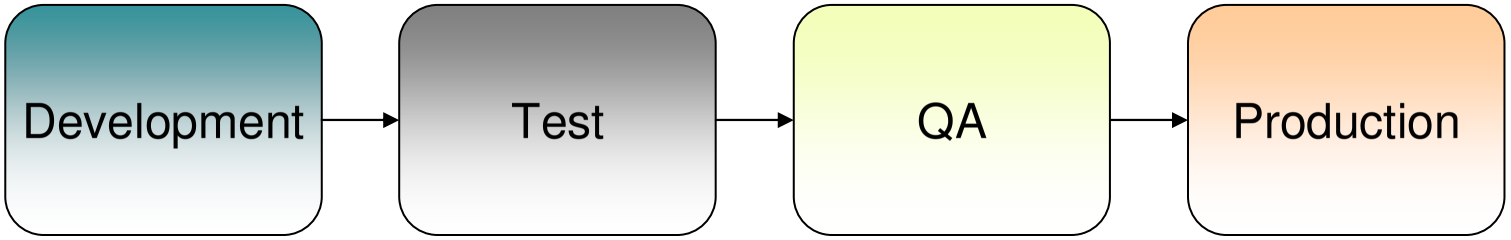
- The Eclipse Platform provides standard ways to build projects, perform version control, and provide for the development of custom plug-ins, such as resource editors to allow users to create project resources easily. Examples are custom editors to aid flow creation, ESQL editing and syntax checking, message set modelling, and a raft of other activities.

- The brokers tooling is created as Eclipse plug-ins, so if you're already using other products that uses Eclipse they will all share the same environment. More than this, resources can be shared between plug-ins, so the fact that the WebSphere Brokers Message Set editor is based on XML schema means it can be shared between any XML schema aware Eclipse plug-in. The tooling is based on Rational Application Developer.

- Broker runtime.
  - Execution groups are container processes which allows message flows to be separated for isolation and scalability. Inside an execution group, the flow is a runtime instantiation (code!) of the model deployed from the Message Broker Toolkit. It uses the WebSphere Message Broker runtime libraries and node libraries to perform the node operations defined and deployed.
  - Each message flow has a pool of threads in an execution group process. Message flow input nodes have a thread assigned from the pool and additional threads move to and from the pool as the message arrival rate rises and falls. It is possible to configure some input nodes to use their own additional pool of threads.
  - Message Sets are typically stored in a database and instantiated with the first message, according to the parser's design.
  - The broker runtime runs on many different platforms, and requires a single, local MQ queue manager. This is used for internal communication and communication with the broker.
- Broker Explorer
  - This tool is based on the WebSphere MQ Explorer and allows administrators to manage queue managers and brokers from the same perspective, and also view Broker performance information, offload WS-Security onto DataPower appliances, and more.
  - For information on all the components inside WebSphere Message Broker, see the component overview documentation here: http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ab20551_.htm

Support for All User Roles and Environments

- Application Developer
  - Develops message flows, message models etc.
  - Unit Tests on local machine
  - Creates Broker Archive (BAR) files containing required artefacts
- Administrator
  - Customizes BAR for target environment (message flow properties including queues, database names etc.)
  - Deploys BAR to target broker
  - Broker management and operational control
  - Monitoring



- WebSphere Message Broker recognizes two typical user roles:

- The Application Developer – responsible for developing message flows and related artefacts
- The Broker Administrator – responsible for installation, configuration and maintenance of broker domains.

- Typically the Application Developer will not be aware of the environment to which message flows will be deployed; this is the responsibility of the broker administrator. Similarly, the broker administrator may not be aware of the exact message flow logic supplied by the application developer. WebSphere Message Broker provides tools and techniques to enable Message Broker “applications” to easily flow between the application developer and administrator.

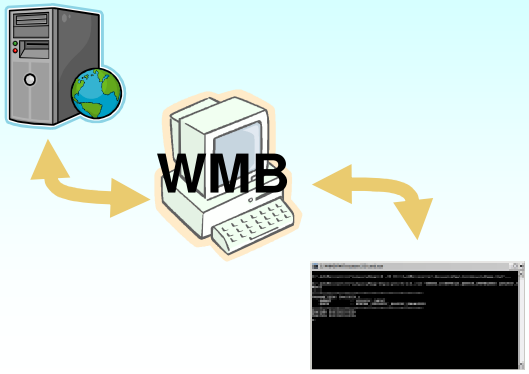
- Additionally, enterprises using Message Broker tend to have several distinct environments of brokers that represent the development lifecycle of a Message Broker solution.

- Development: Developers of message flows and related artefacts will tend to have a development or Unit Test environment on their own system.
- Test: Usually on a system or set of systems remote to the developers, test domains are used for functional verification or system testing of broker message flows.
- QA: Usually a mirror of the Production domain, used for final testing of message brokers before they are promoted.
- Production: The “live” Message Broker system that is providing value to the business.

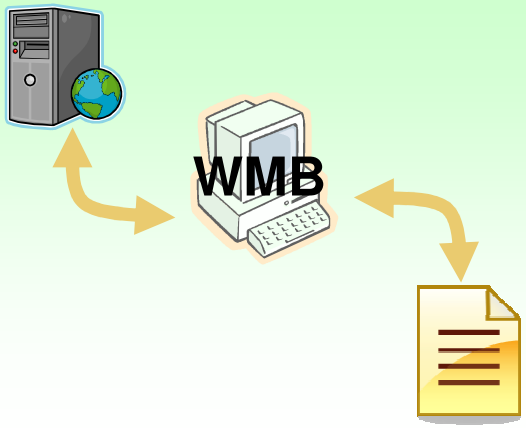
Key Usage Scenarios

SHARE
Technology • Connections • Results

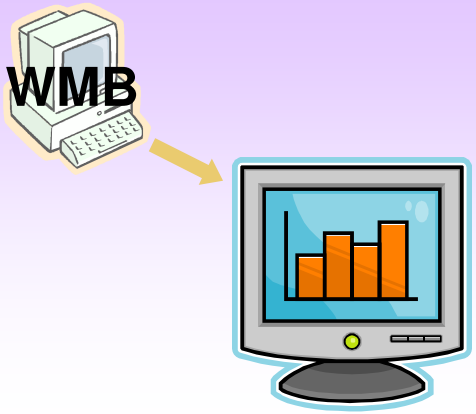
Extending the reach of existing applications



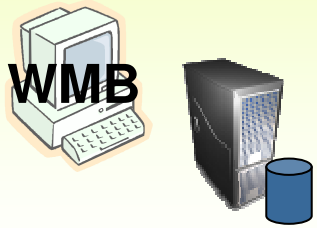
Moving Batch Into Online



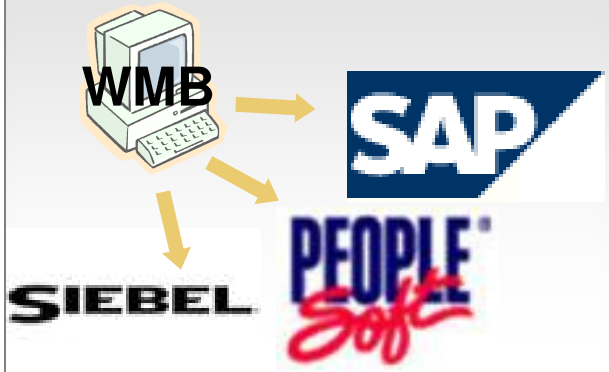
Business Monitoring



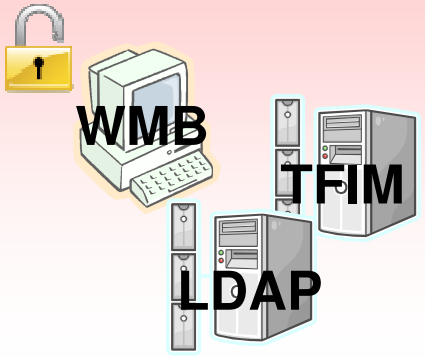
Making an Application Inventory and Governing Processing



Making the Most of Packaged Applications

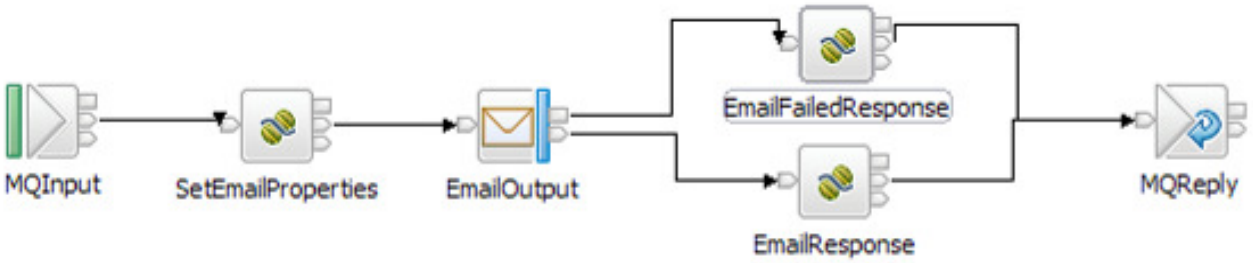


Participating in a Secure Infrastructure

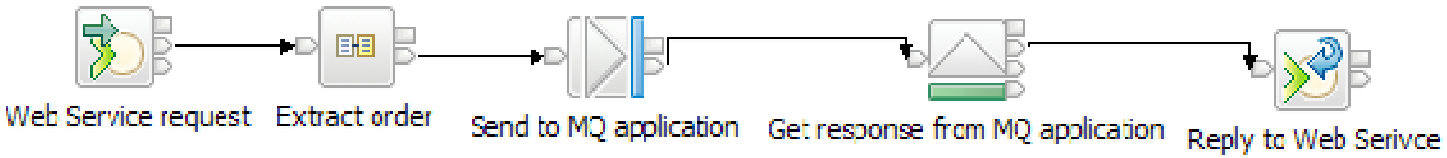


- Key usage scenarios

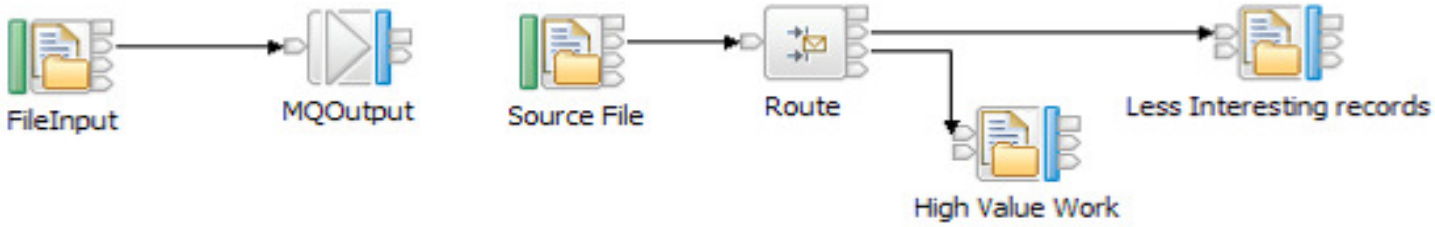
- Usage Scenario 1: Extending the Reach of Existing Applications
  - MQ enable all your applications!
  - Many enterprises have realized the advantages of MQ technology: Robust, transactional, reliable, high-performance messaging
  - Message Broker allows you to use MQ technology to the fullest extent. Message Broker has an incredibly broad range of connectivity mechanisms, and any application can easily connect to the MQ infrastructure inbound or outbound.
  - For example, transform TCP/IP based application to generate regular MQ messages. Or existing MQ based applications can connect to other applications using MQ.



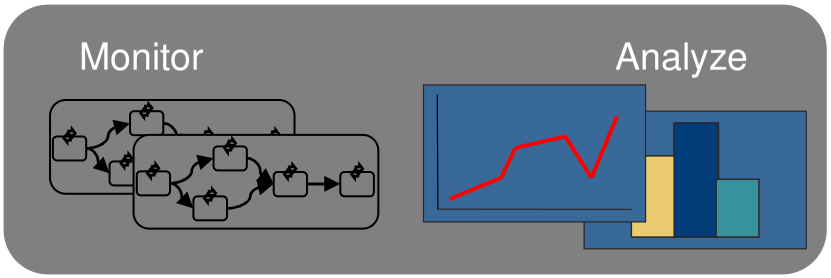
- Usage Scenario 1: Extending the Reach of Existing Applications (cont)
  - Provide and Consume Web Services!
  - Web services are now established as an interoperability standard. They are vitally important from a business to business connectivity perspective, and it is now possible for businesses to consume each others services using these well defined standards. Web services allow internal standardization between different parts of the same organization
  - However, the adoption of Web Services by many subsystems is NOT universal. Message Broker can be used a universal translator to convert between web service and existing formats and protocols. This allows your existing applications to be exposed as web service, and existing applications can consume web services without change. Developers can use web services without new development skills or platform knowledge.



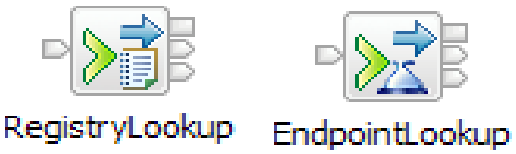
- Usage Scenario 2: Combining File-based and Online Processing
  - Get first class data from your files!
  - Files exchange between applications still popular and effective. It's a flexible method of exchange: Neither enterprise has to mandate technology. There are legitimate reasons for using files to exchange information, and these usually relate to the way businesses run or physical processes occur.
  - For example, a cargo ship has thousands of containers each with hundreds of palettes, and files can reduce unit transaction costs by aggregating numerous clients requests.
  - Message Broker's file processing allows clients to get file/batch work online, easily. Flat-file nodes can handle huge files (e.g. gigabytes) without storage growth (including FTP). There are also z/OS specific VSAM file nodes (Input, Read, Write, Update, Delete).



- Usage Scenario 3: Business Activity Monitoring
  - Understand the importance of ESB data!
  - Message Broker allows emission and processing of events from many sources and targets. For example, the capture business relevant information (such as total dollar trade value per day, or total number of orders per hour) to feed to WebSphere Business Monitor.
  - Message Broker can generate monitoring events from message flows. This enables integration with WebSphere Monitor and Modeller to monitor and analyze KPIs. It's a non-invasive solution, meaning that the administrator can operationally activate events on new and existing message flows.
  - Event emission occurs through standard MQ publish/subscribe, meaning that any applications can consume the event, local or remote. There can also Business Monitor can be local or remote, and there can multiple, concurrent listeners to events.

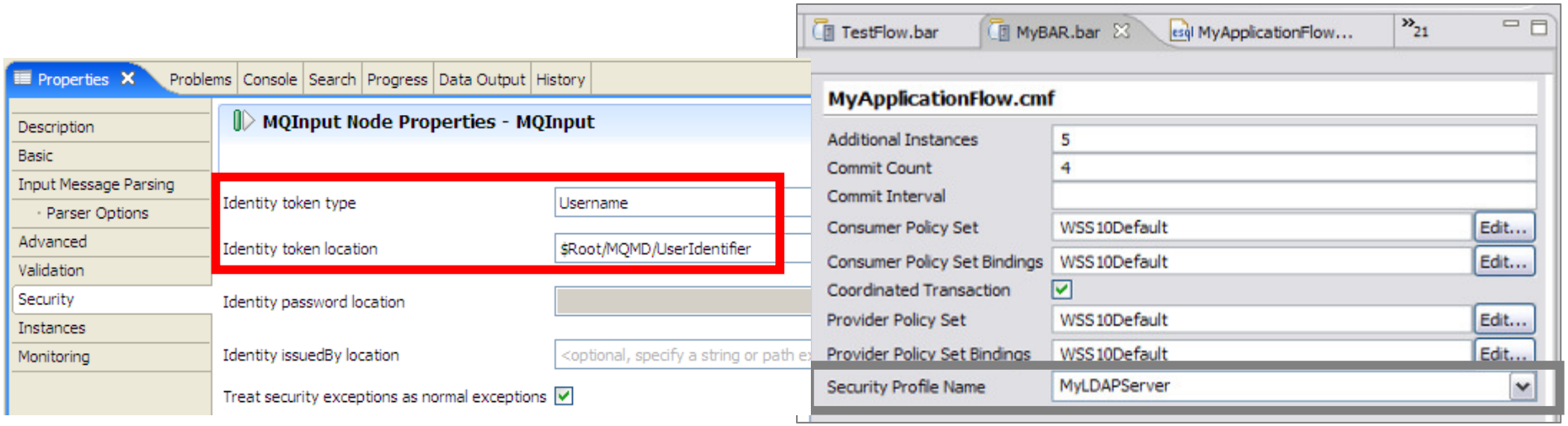


- Usage Scenario 4: Make an Application Inventory and Get Value From It
  - Understand your application assets and control their access dynamically!
  - A great first step to SOA is to catalog application and service assets in a registry, such as WSRR.
  - Classify your services by function, owning department or some other metric. Build relationships beterrn services for lifecycle management and versioning. Store metadata for services, such as user-qualified properties (for example, UserClass=GOLD or UserClass=SILVER).
  - You can use WSRR registry information from within Message Broker. There are built-in nodes that allow message flows to access the registry. Primary use cases are governance (who can access which applications and services), and dynamicity (update registry to change connectivity behaviour). The nodes use an in-memory cache for high performance, which also includes cache invalidation options.
  - Also check out the MQ service definition to describe MQ applications as services and store them. Access these in Message Broker using the RegistryLookup node.

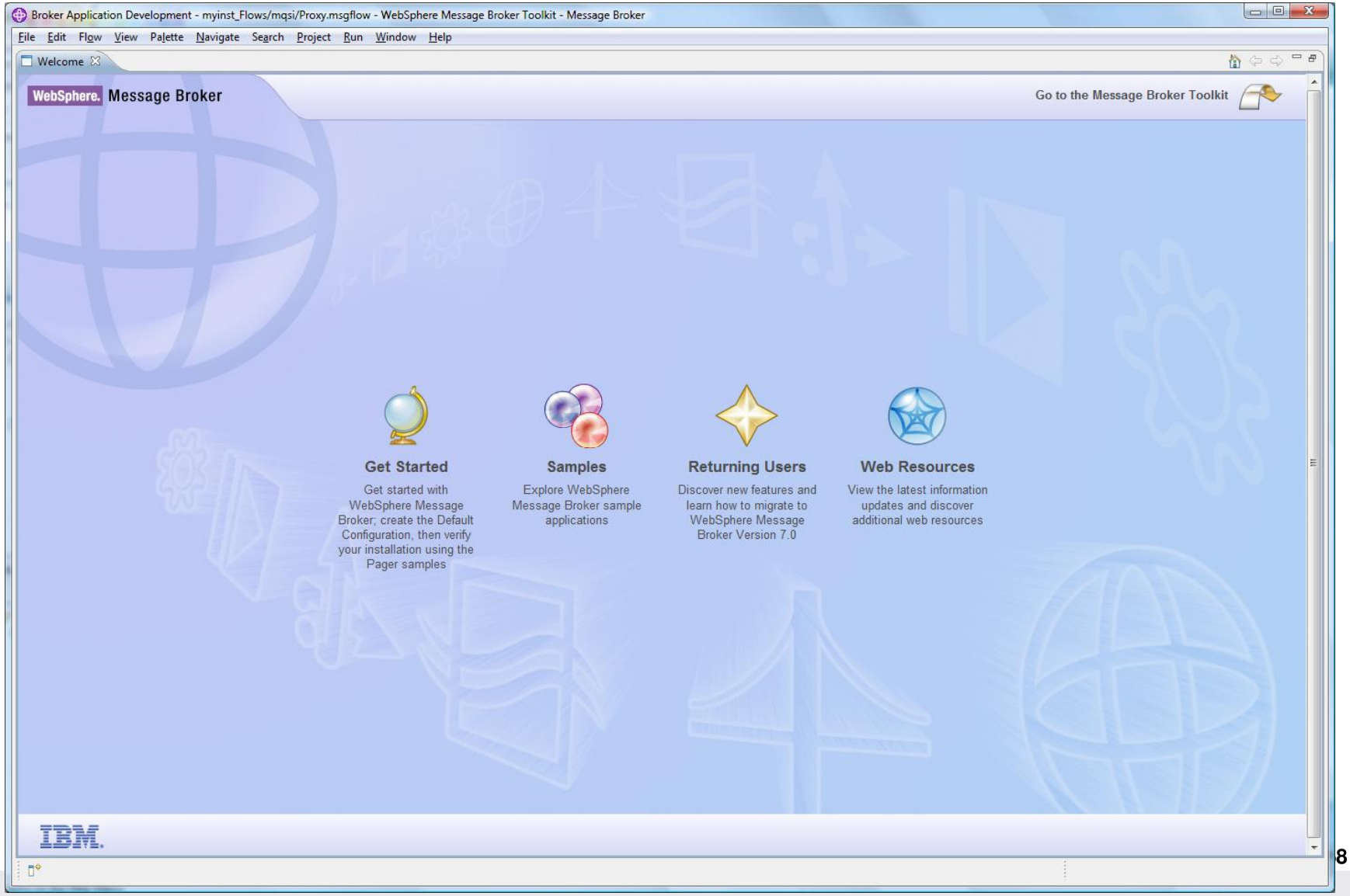


- Usage Scenario 5: New Work To and From Packaged Applications
  - Move information to and from packaged applications!
  - Packaged applications play a vital role in many businesses: SAP for purchasing, sales, inventory, SEBL for sales and PeopleSoft for HR.
  - Interfaces are often non standard: e.g. SAP BAPIs, IDOCs. Consequently, processing and data are isolated from other applications, which means that packaged applications have difficultly using and generating information for other applications. This inhibits adoption of a best of breed philosophy.
  - Message Broker has built-in nodes support for SAP, SEBL, PeopleSoft, inbound and outbound. This allows you to drive new work into its packaged application from any other MB connection. You can send information from packaged application to any other MB connection.
  - This means that packaged applications can focus on what they do best and be integrated.

- Usage Scenario 6: Participate in a Secure Infrastructure
  - Secure applications with identity, authentication and authorization!
  - Application connectivity scenarios imply security domain changes. Identity management, authorization, and authentication are mechanisms essential to providing proper participation in a secure infrastructure.
  - Message Broker can perform a key role as a Policy Enforcement Point (PEP), collaborating with using Policy Decision Point (PDP) such as an LDAP or Tivoli Federated Identity Manager (TFIM) server.



Demo


Broker Application Development - myinst_Flows/mqsi/Proxy.msgflow - WebSphere Message Broker Toolkit - Message Broker
File Edit Flow View Palette Navigate Search Project Run Window Help
Welcome
WebSphere. Message Broker
Go to the Message Broker Toolkit
Get Started
Get started with WebSphere Message Broker; create the Default Configuration, then verify your installation using the Pager samples
Samples
Explore WebSphere Message Broker sample applications
Returning Users
Discover new features and learn how to migrate to WebSphere Message Broker Version 7.0
Web Resources
View the latest information updates and discover additional web resources
IBM.

- This slide shows the welcome screen that is shown when you start the Message Broker Toolkit for the first time.
- It is the starting point for the demo… assuming we have time!

WebSphere Message Broker

- Universal Connectivity
  - Simplify application connectivity to provide a flexible and dynamic infrastructure
- Routes and \transforms messages FROM anywhere, TO anywhere
  - Supports a wide range of protocols
    - MQ, JMS 1.1, HTTP(S), Web Services, File, EIS (SAP,SEBL…), TCP/IP, User Defined
  - Supports a broad range of data formats
    - Binary (C/COBOL), XML, Industry (SWIFT, EDI, HIPAA…), User Defined
  - Interactions and Operations
    - Route, Filter, Transform, Enrich, Monitor, Distribute, Decompose, Correlate, Detect…
- Simple programming
  - Patterns based for top-down, parameterized connectivity of common use cases
    - Web Service façades, message oriented processing, queue to file…
  - Construction based for bottom-up assembly of bespoke connectivity logic
    - Message flows to describe application connectivity comprising…
    - Message nodes which encapsulate required integration logic which operate on…
    - Message tree which describes the data in a format independent manner
  - Transformation options include graphical mapping, PHP, Java, ESQL, XSL and WTX
- Operational Management and Performance
  - Extensive Administration and Systems Management facilities for developed solutions
  - Wide range of operating system and hardware platforms supported
  - Offers performance of traditional transaction processing environments

- This is a single slide that describes what WebSphere Message Broker is and the main concepts that have been introduced.

- I hope you have enjoyed this technical introduction to WebSphere Message Broker.

Copyright and Trademarks