IBM Software Group

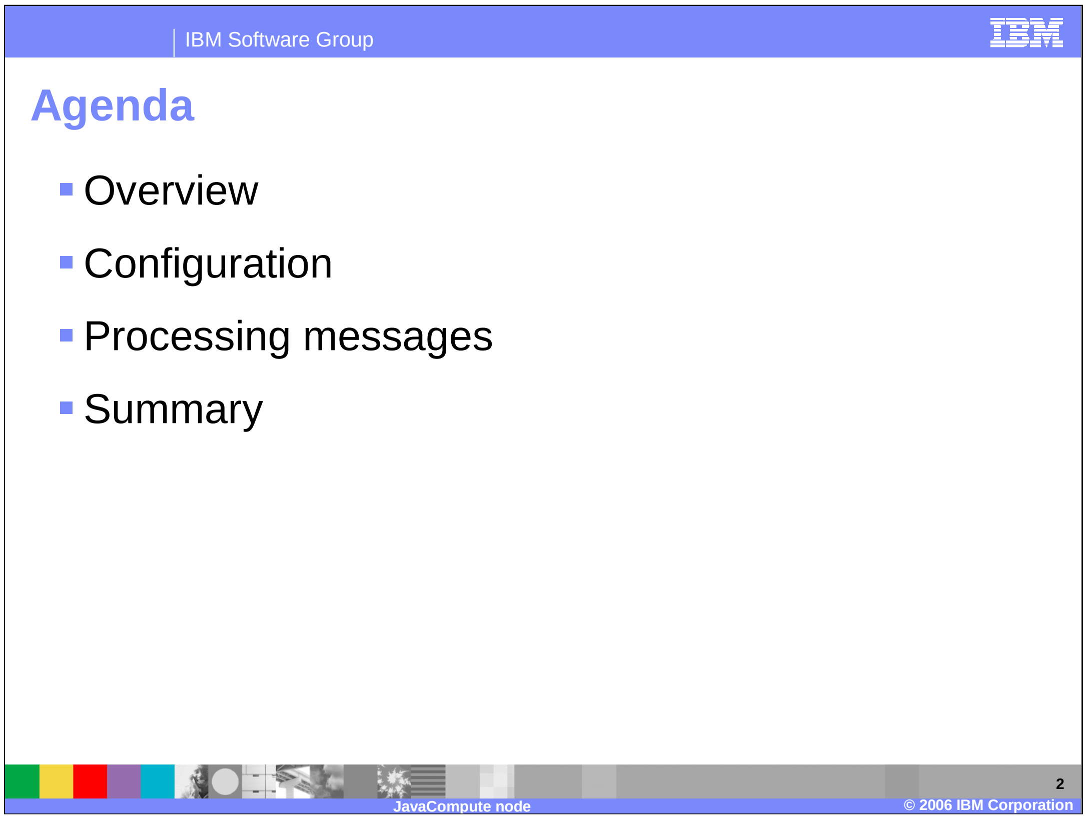
# WebSphere® Message Broker Version 6

## *JavaCompute node*

business on demand.

Updated December 19, 2006

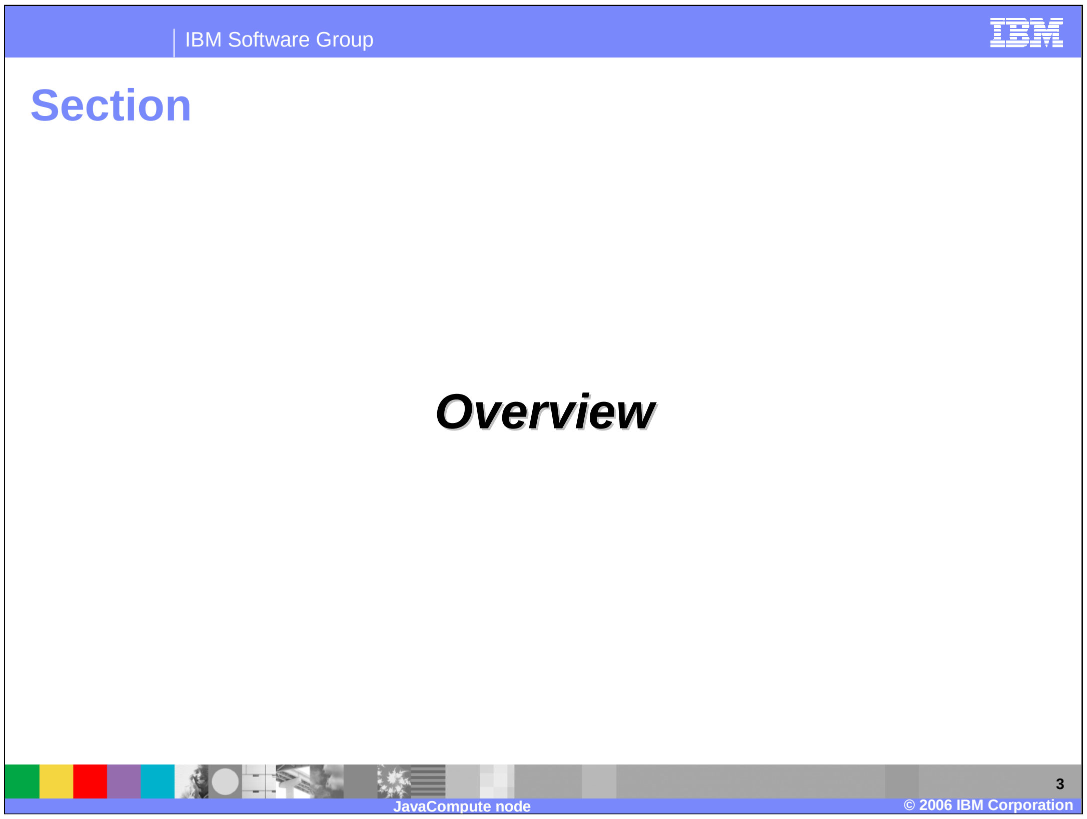
This session looks at the new JavaCompute node, introduced in Message Broker Version 6.



## Agenda

- Overview
- Configuration
- Processing messages
- Summary

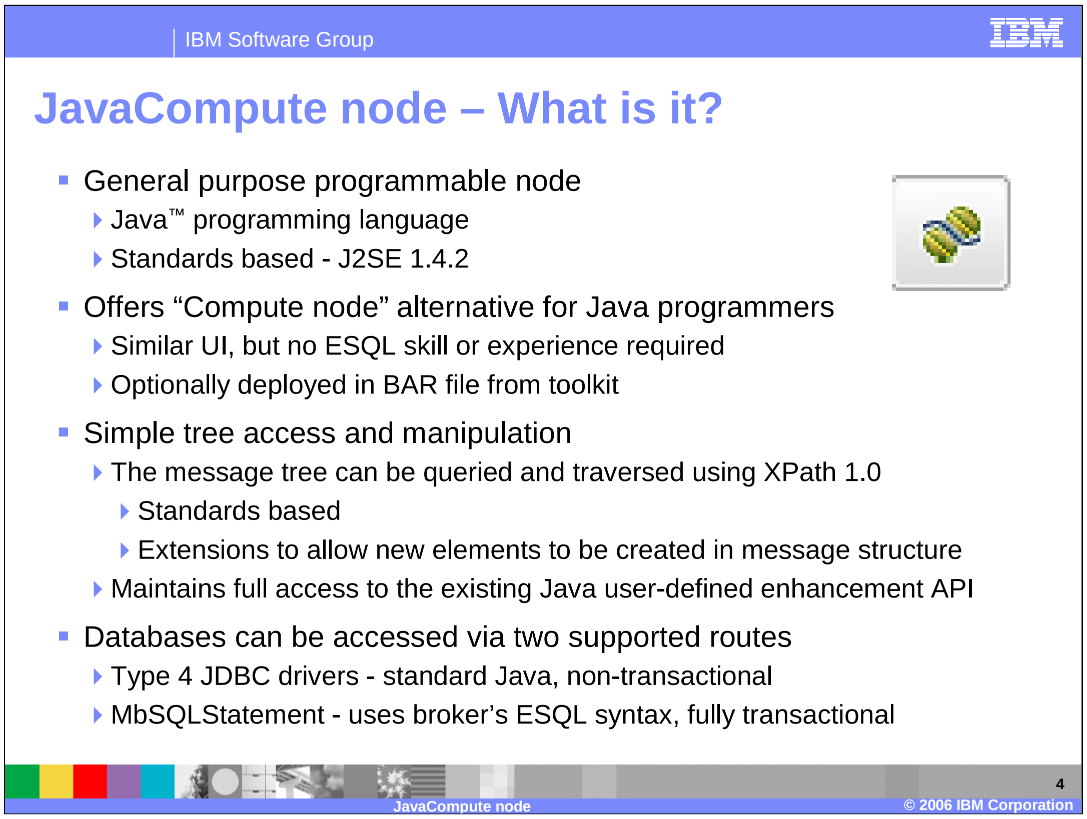
The agenda for this presentation includes an overview, configuring, and processing messages.



## Section

**_Overview_**

This section will overview the JavaCompute node.


IBM Software Group

## JavaCompute node – What is it?

- General purpose programmable node
  - Java™ programming language
  - Standards based - J2SE 1.4.2
- Offers "Compute node" alternative for Java programmers
  - Similar UI, but no ESQL skill or experience required
  - Optionally deployed in BAR file from toolkit
- Simple tree access and manipulation
  - The message tree can be queried and traversed using XPath 1.0
    - Standards based
    - Extensions to allow new elements to be created in message structure
  - Maintains full access to the existing Java user-defined enhancement API
- Databases can be accessed via two supported routes
  - Type 4 JDBC drivers - standard Java, non-transactional
  - MbSQLStatement - uses broker's ESQL syntax, fully transactional

4

JavaCompute node © 2006 IBM Corporation

WebSphere Message Broker Version 6 has introduced the capability for message flow compute nodes to be written entirely in Java. In previous versions, processing logic had to be written in ESQL, although Version 5 of the broker did allow the possibility of calling a Java routine from within an ESQL Compute node.

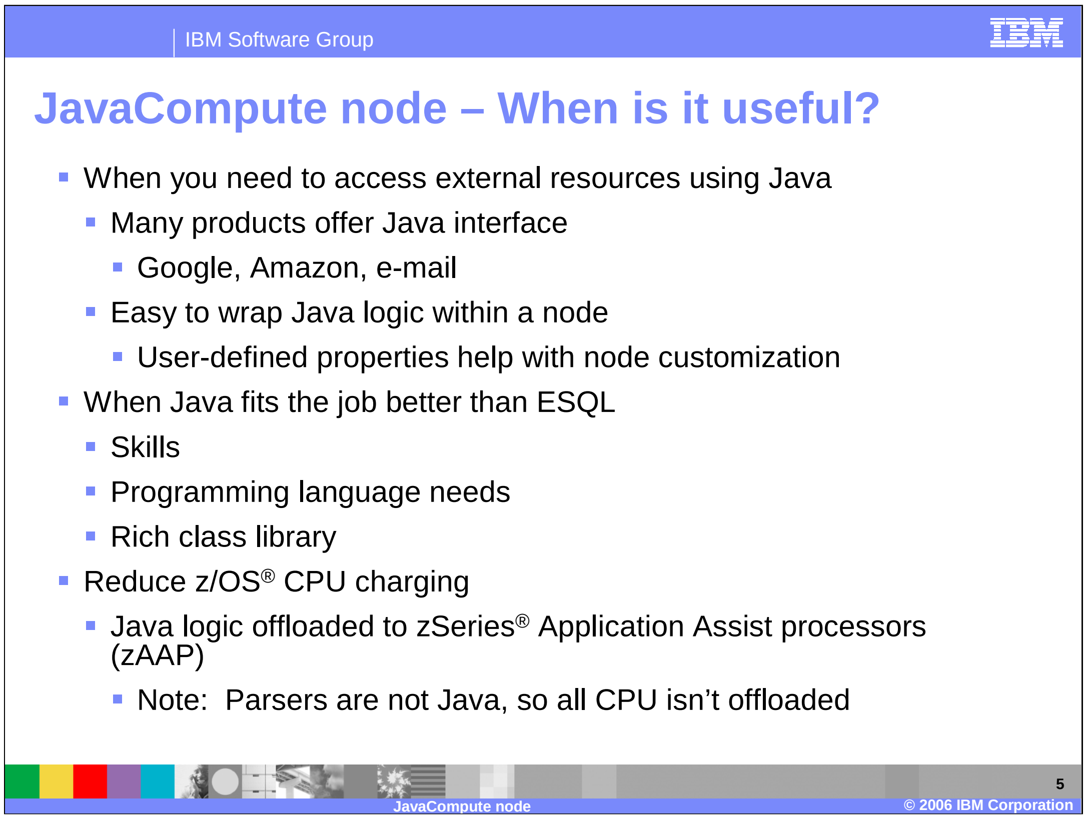
The new node provides a full Java implementation, and is based on the Java 1.4.2 level. The Java code that is used by the node is stored in an Eclipse Java project. The JavaCompute node has the same look and feel as other nodes, with similar properties and connecting terminals. The resulting jar file is deployed to the broker by including it in the broker archive file.

Message trees can be accessed from within the JavaCompute node. If required, this can be done using the XPath tools, similar to that used within the Mapping node.  However, there is no 'content-assist' to complete an XPath like there is in ESQL.

If you have developed Java user-defined enhancements using the existing mechanism, the Java API is still maintained.

Relational databases can be accessed in two ways. Type 4 JDBC drivers can be used from within the JavaCompute node. However, this is currently a non-transactional interface, so updates will not be coordinated with any other updates done within the same message flow, including any updates to MQ.

Alternatively, databases can be updated using the MbSQLStatement class. This provides full transactional integrity across all updated resources.



## JavaCompute node – When is it useful?

- When you need to access external resources using Java
  - Many products offer Java interface
    - Google, Amazon, e-mail
  - Easy to wrap Java logic within a node
    - User-defined properties help with node customization
- When Java fits the job better than ESQL
  - Skills
  - Programming language needs
  - Rich class library
- Reduce z/OS® CPU charging
  - Java logic offloaded to zSeries® Application Assist processors (zAAP)
    - Note: Parsers are not Java, so all CPU isn't offloaded

The JavaCompute node can be used in three ways:

1) to look at a message, and propagate the message to an output terminal based on the message content. The message content is not changed and you have read-only access to the message.

2) to modify a message, and propagate the modified message to an output terminal.

3) to create a new message, and propagate the new message to an output terminal.

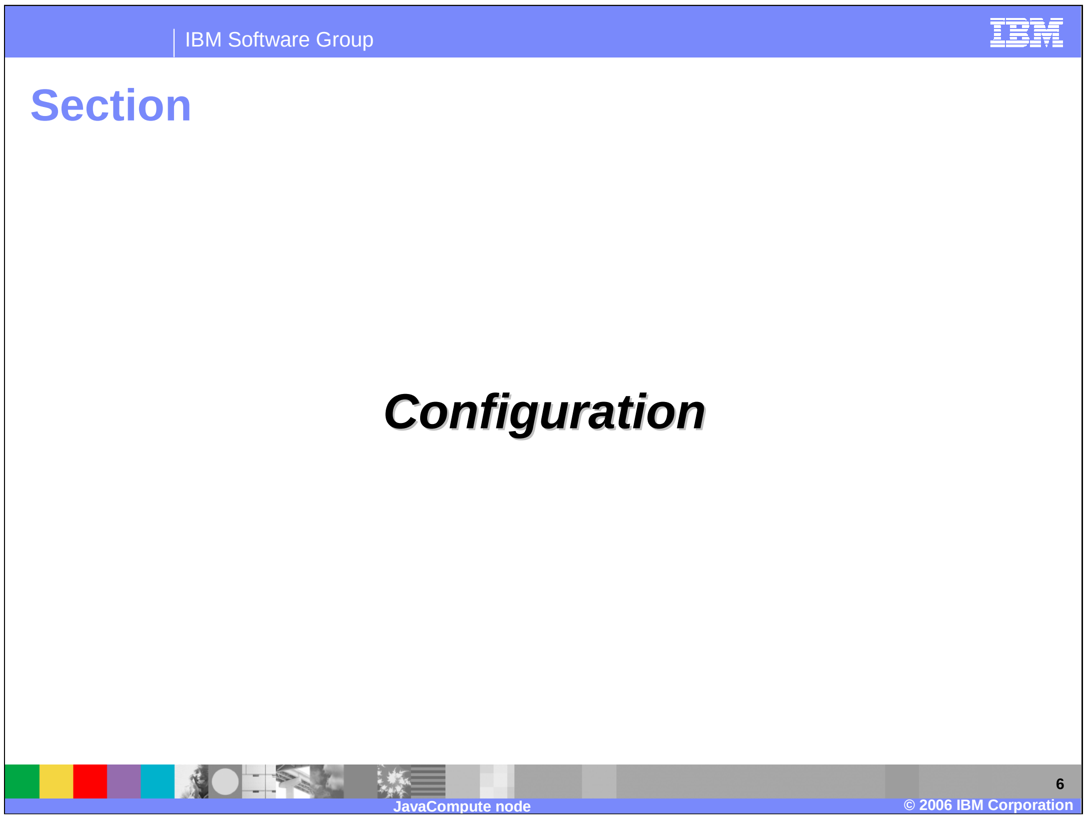
The JavaCompute node can be used to do many things. The previous three options are presented in a wizard when creating a new JavaCompute node. There are a very rich set of Java libraries available to this node, providing interfaces to external products.

Many application interfaces are compatible only with Java. Services such as Google and Amazon offer this type of interface, and if web services are not well established in your environment, it may be easier to call such services directly via Java. Function isolation is possible within a JavaCompute node by passing user-defined properties.

One of the samples that is shipped with message broker Version 6 is an implementation of the Google Java API.

The JavaCompute node provides an alternative to ESQL, allowing use of existing Java skills and reuse of pre-built Java code.

On z/OS, it is possible to reduce the overall CPU usage by taking advantages of the zAAP charging mechanism. Using this, all Java processing can be offloaded to a separate processing engine, which does not incur software charges.


IBM Software Group

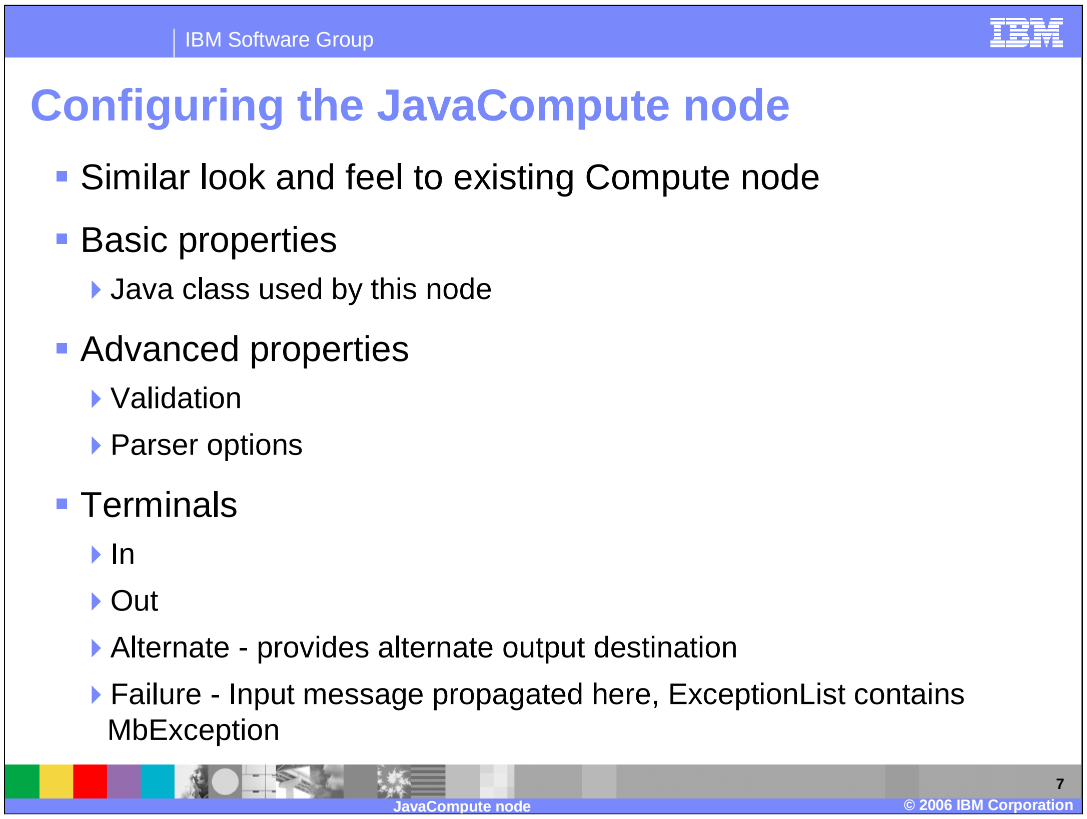
## Section

*Configuration*

6

JavaCompute node

This section covers configuration.


IBM Software Group

## Configuring the JavaCompute node

- Similar look and feel to existing Compute node
- Basic properties
  - Java class used by this node
- Advanced properties
  - Validation
  - Parser options
- Terminals
  - In
  - Out
  - Alternate - provides alternate output destination
  - Failure - Input message propagated here, ExceptionList contains MbException

7

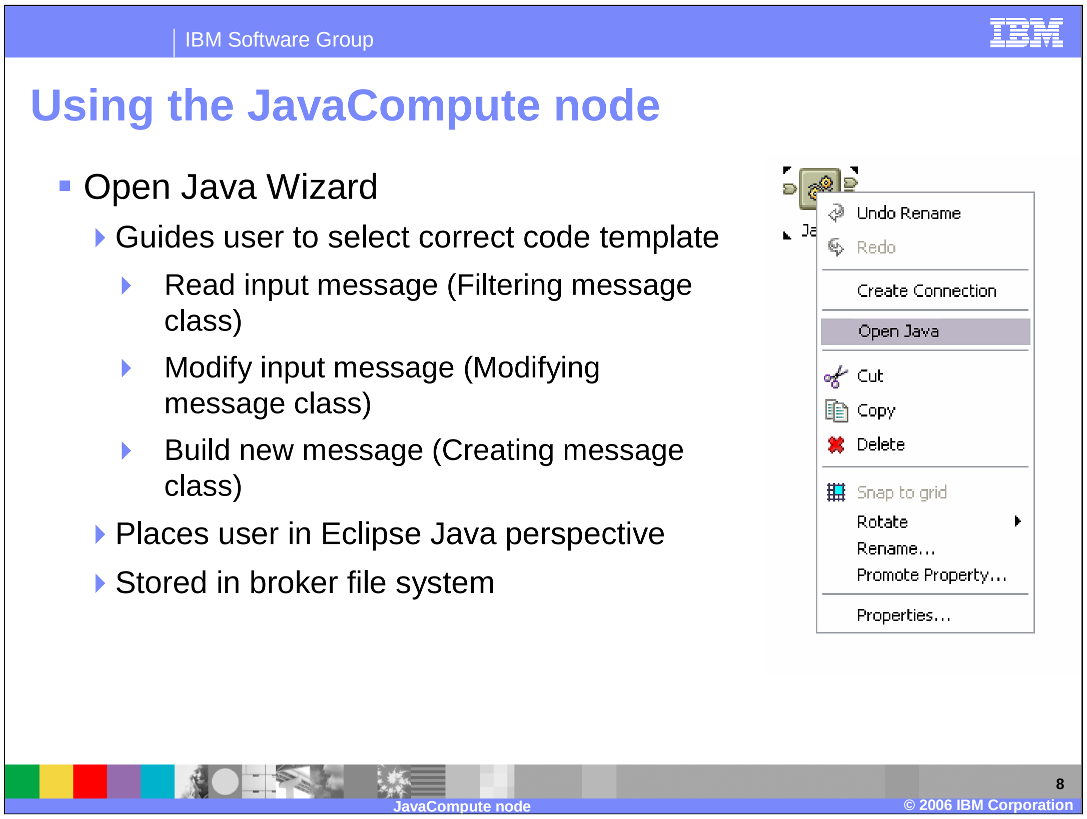
JavaCompute node

© 2006 IBM Corporation

Using the JavaCompute node in a message flow follows the usual node conventions. The node has a similar feel to the existing Compute node.

In the Properties of the node, the Basic and Advanced sections allow you to specify parameters. The JavaCompute node has only one basic property. This is *Java Class*. Enter the name of the Java class that is used in this node. This name must be in the list of JavaCompute node classes that are available in the project references for the message flow project.  This is the only property which must be set for the node.  The Advanced properties provide options for validation and parsing.

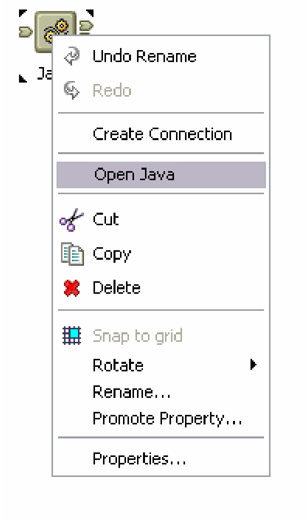
The node has one input terminal and three output terminals. The output terminals are the standard failure terminal and two general purpose output terminals. This enables the node to be used as a general purpose routing node.



IBM Software Group

## Using the JavaCompute node

- Open Java Wizard
  - Guides user to select correct code template
    - Read input message (Filtering message class)
    - Modify input message (Modifying message class)
    - Build new message (Creating message class)
  - Places user in Eclipse Java perspective
  - Stored in broker file system



8

JavaCompute node

© 2006 IBM Corporation

You configure each instance of the JavaCompute node that occurs in a message flow.  To do this, right-click the node in the editor view. The Open Java option is presented. Select this option (as shown in this slide).

The first time that you select Open Java for a node, a wizard is launched that guides you through the creation of a new Java project and a Java class that contains skeleton code. You are given a selection of templates to use for this node; the options are:

Read input message

Modify input message

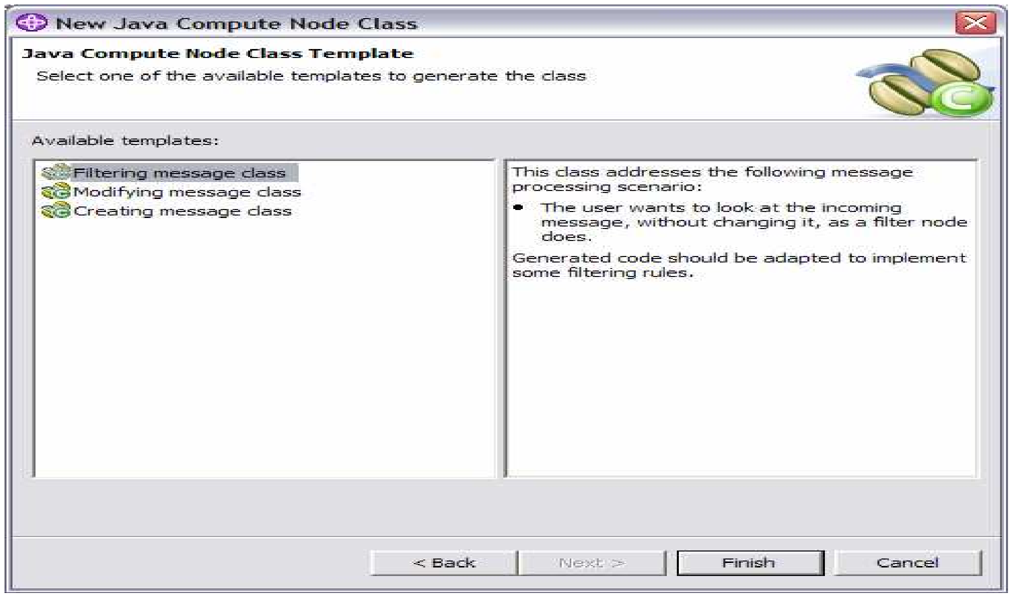
Create new message

This skeleton code is presented in a Java editor.

If this is not the first time that you have done this, you are presented with the Java code that you have created for this node in a Java compute perspective.

Java files are managed through the Java perspective and stored in the toolkit file system.

## Using the JavaCompute node



Java Compute Node Class Template

Select one of the available templates to generate the class

Available templates:

- Filtering message class
- Modifying message class
- Creating message class

This class addresses the following message processing scenario:

- The user wants to look at the incoming message, without changing it, as a filter node does.

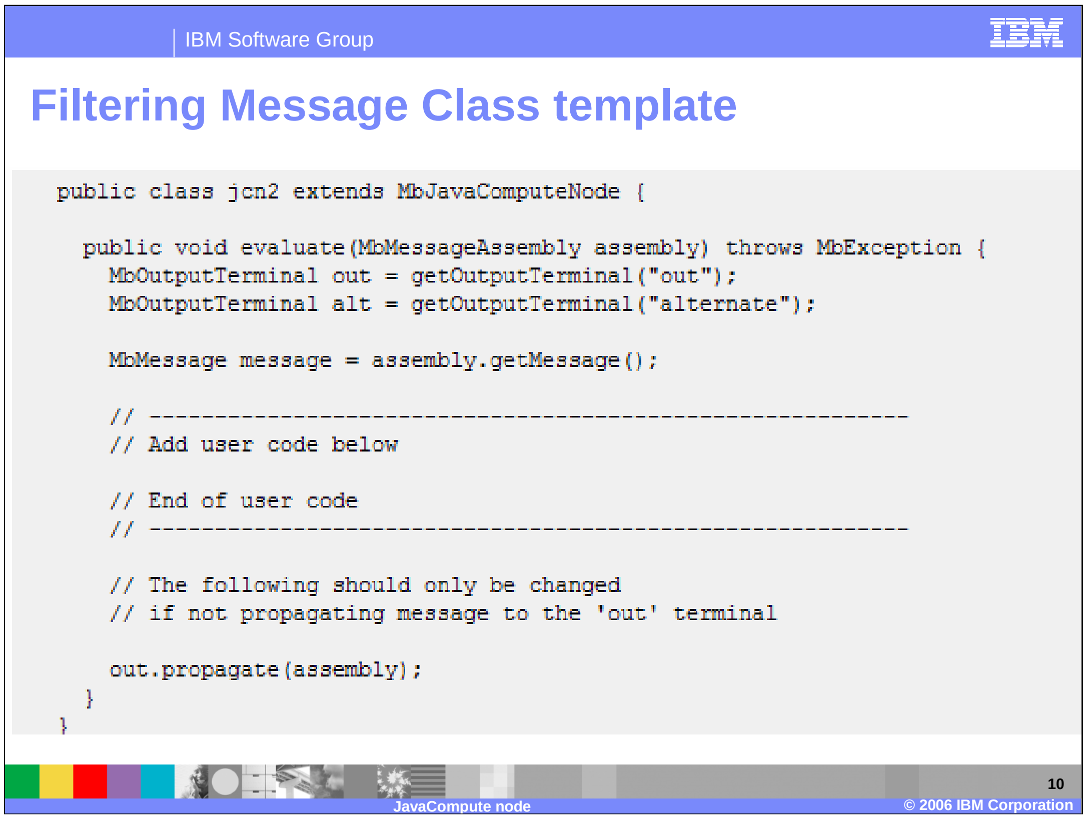
Generated code should be adapted to implement some filtering rules.

This screen capture shows the part of the wizard where you select the Class that provides the functions and templates for your processing needs:

Filtering (read input message)

Modifying input message

Creating new message


IBM Software Group

## Filtering Message Class template

```
public class jcn2 extends MbJavaComputeNode {

  public void evaluate(MbMessageAssembly assembly) throws MbException {
    MbOutputTerminal out = getOutputTerminal("out");
    MbOutputTerminal alt = getOutputTerminal("alternate");

    MbMessage message = assembly.getMessage();

    // ----------------------------------------------------------
    // Add user code below

    // End of user code
    // ----------------------------------------------------------

    // The following should only be changed
    // if not propagating message to the 'out' terminal

    out.propagate(assembly);
  }
}
```

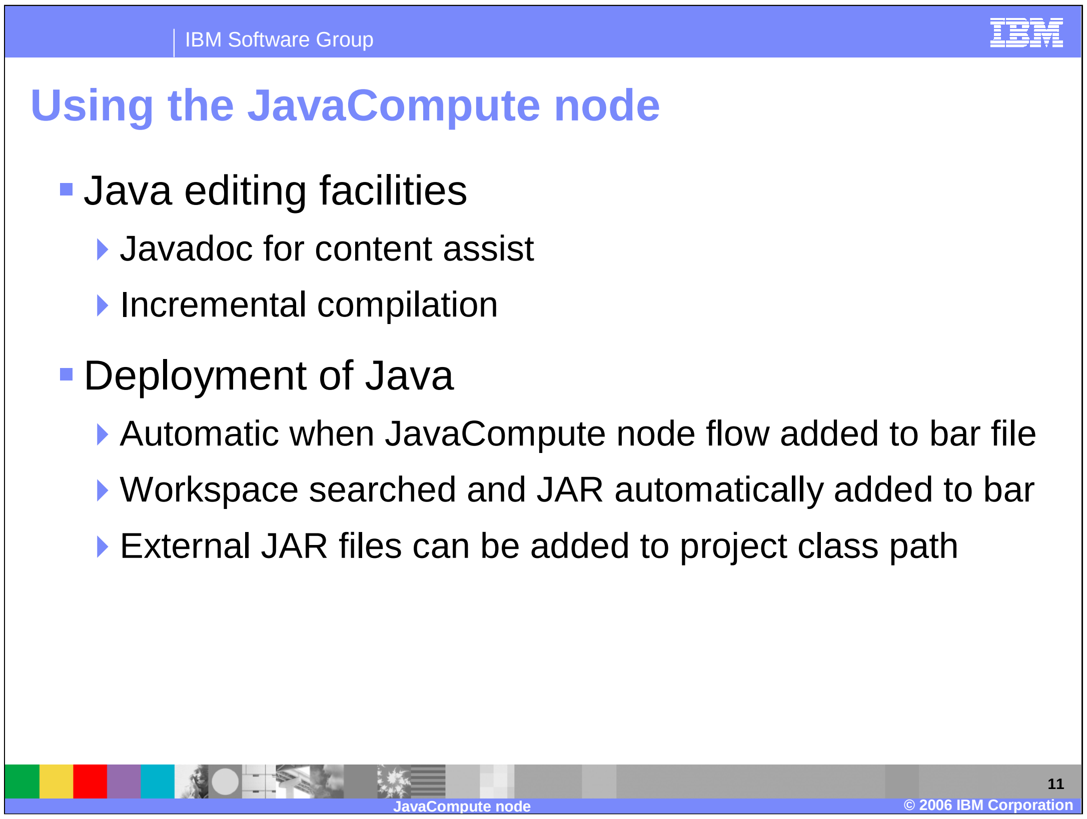
10

JavaCompute node

© 2006 IBM Corporation

When you select the *Filtering Message Class* template in the JavaCompute node creation wizard, template code is produced. The code shown in this slide passes the input message to the Out terminal without doing any processing on the message. The template produces a partial implementation of a method called evaluate(). The broker calls evaluate() once for each message that passes through the node. The parameter that is passed to evaluate() is the message assembly. The message assembly encapsulates the message that is passed on from the previous node in the message flow.

You can add custom code to the template to propagate messages to both the Out and Alternate terminals to create a message filter.



## Using the JavaCompute node

- Java editing facilities
  - Javadoc for content assist
  - Incremental compilation
- Deployment of Java
  - Automatic when JavaCompute node flow added to bar file
  - Workspace searched and JAR automatically added to bar
  - External JAR files can be added to project class path

You can add any valid Java code to a JavaCompute node, making full use of the existing Java user-defined node API to process an incoming message. You can use the Java editing facilities of the Eclipse platform to develop your Java code. These facilities include:

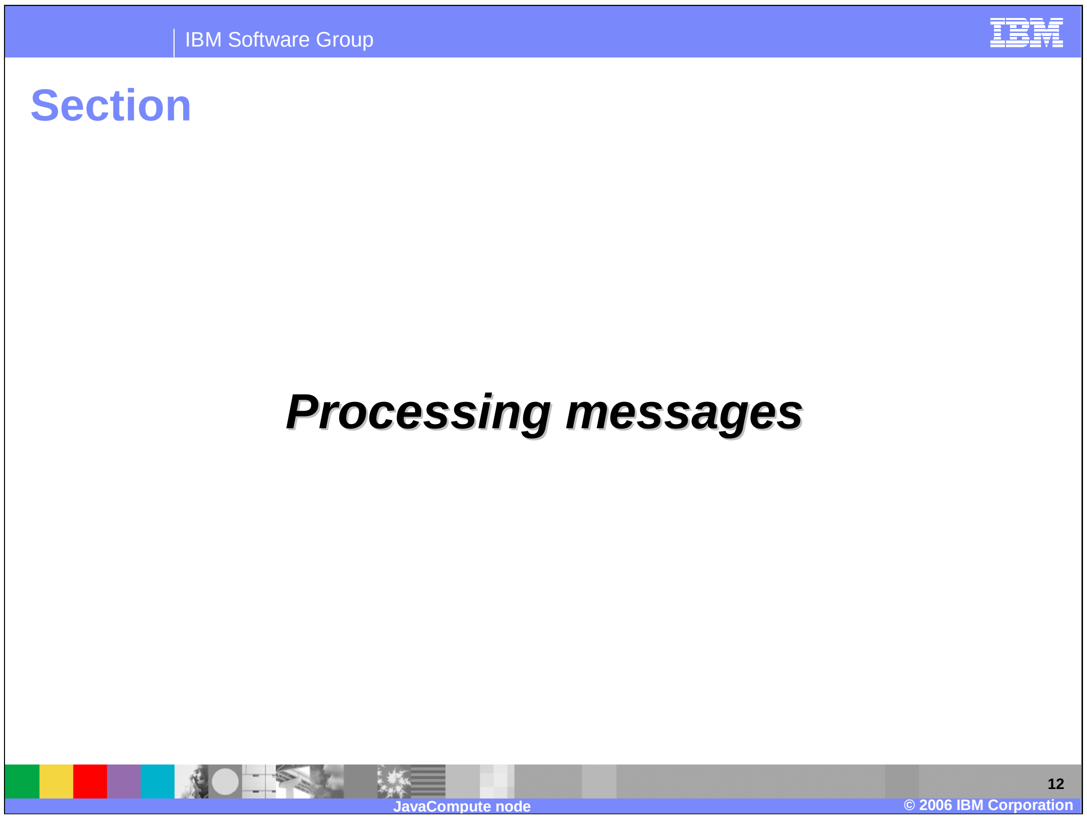
•code completion

•integrated Javadoc documentation

•automatic compilation

The Message Brokers Toolkit handles the deploying of JavaCompute node code automatically. When you create a bar file and add the message flow, the Message Brokers Toolkit packages the compiled Java code (JAR file) and its dependencies into the bar file.

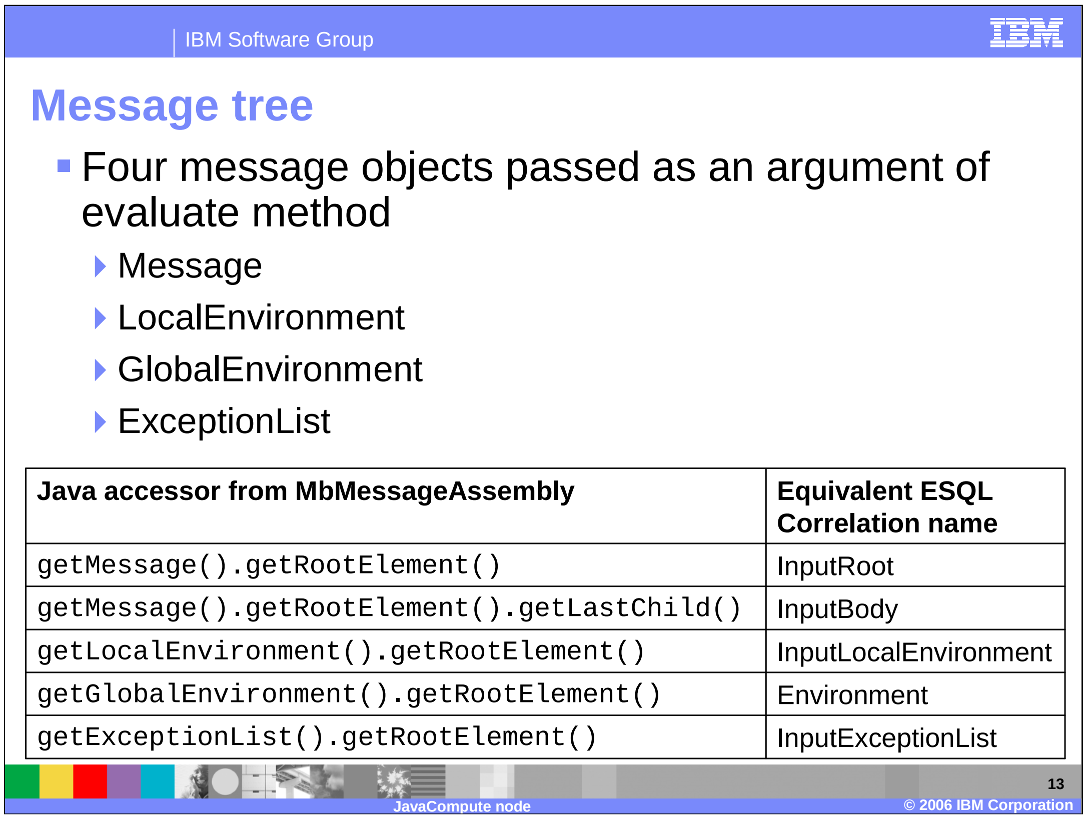
If a message flow has a large JAR file, or has a large number of java dependencies, it is possible for the bar file to become quite large. If this is a concern, it is also possible to manually copy these JAR files to the broker runtime, instead of using the bar file mechanism. When the broker is stopped, you add the directory containing your files to the LILPATH by using the mqsichangebroker command. This must be done for every broker that uses the file.  Details are provided in topic as10004 in the Information Center.



## Section

***Processing messages***

This section covers processing messages.



IBM Software Group

## Message tree

- Four message objects passed as an argument of evaluate method
  - Message
  - LocalEnvironment
  - GlobalEnvironment
  - ExceptionList

| **Java accessor from MbMessageAssembly** | **Equivalent ESQL Correlation name** |
|---|---|
| `getMessage().getRootElement()` | InputRoot |
| `getMessage().getRootElement().getLastChild()` | InputBody |
| `getLocalEnvironment().getRootElement()` | InputLocalEnvironment |
| `getGlobalEnvironment().getRootElement()` | Environment |
| `getExceptionList().getRootElement()` | InputExceptionList |

13

JavaCompute node

© 2006 IBM Corporation

When you want to access the contents of a message, for reading or writing, use the structure and arrangement of the elements in the tree that the parser creates from the input bit stream. Follow the relevant parent and child relationships from the top of the tree downwards, until you reach the required element.

The message tree is passed to a JavaCompute node as an argument of the evaluate method. The argument is a MbMessageAssembly object. MbMessageAssembly contains four message objects:

•Message

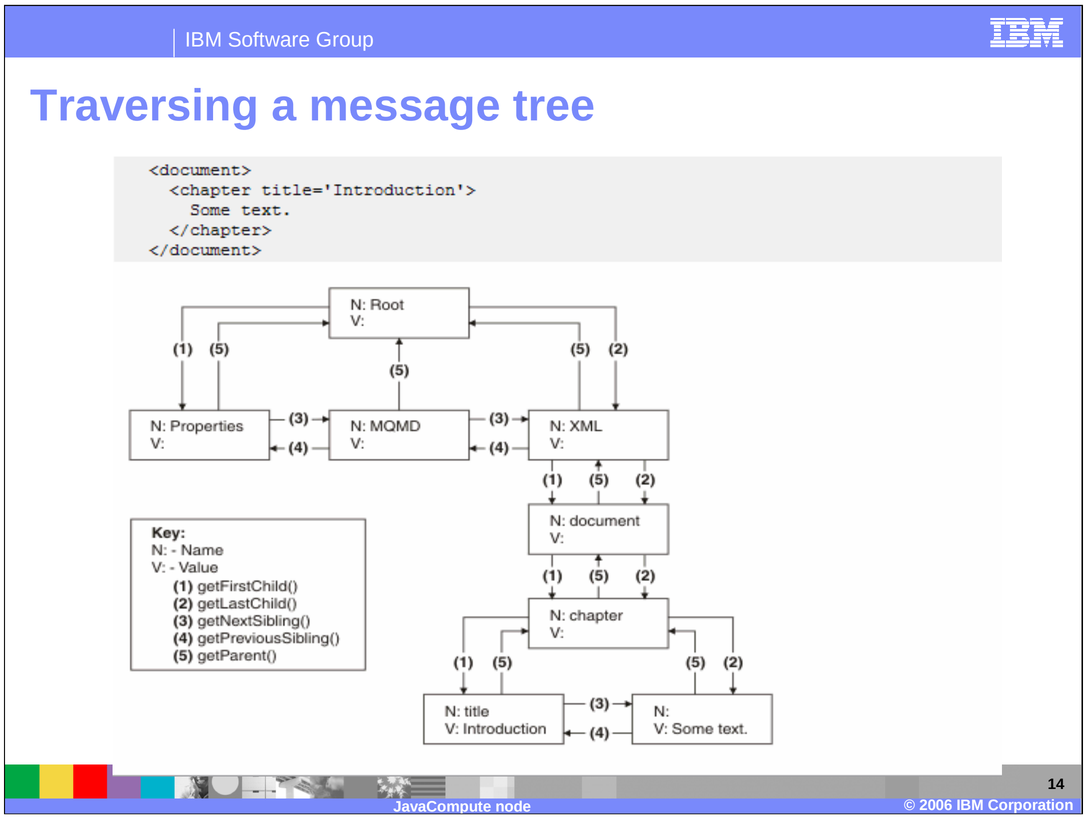
•Local Environment

•Global Environment

•Exception List

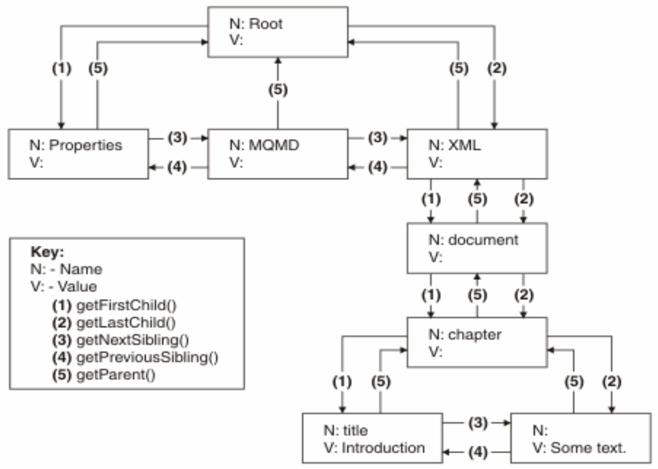
These objects are read-only, except for Global Environment. If you try to write to the read-only objects, a MbReadOnlyException is thrown.

The table at the bottom of this slide shows the corresponding Java methods for the equivalent objects accessible from ESQL statements.



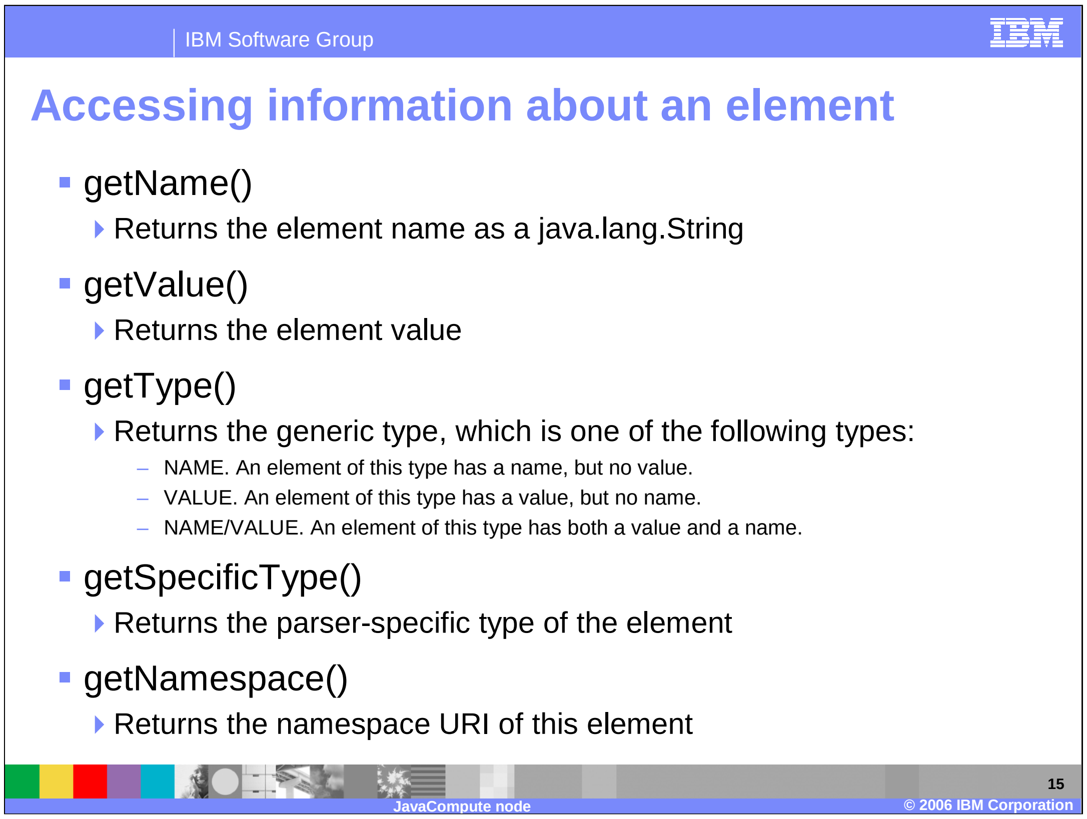
## Traversing a message tree

```
<document>
  <chapter title='Introduction'>
    Some text.
  </chapter>
</document>
```



The JavaCompute node operates on the logical message tree within the broker.

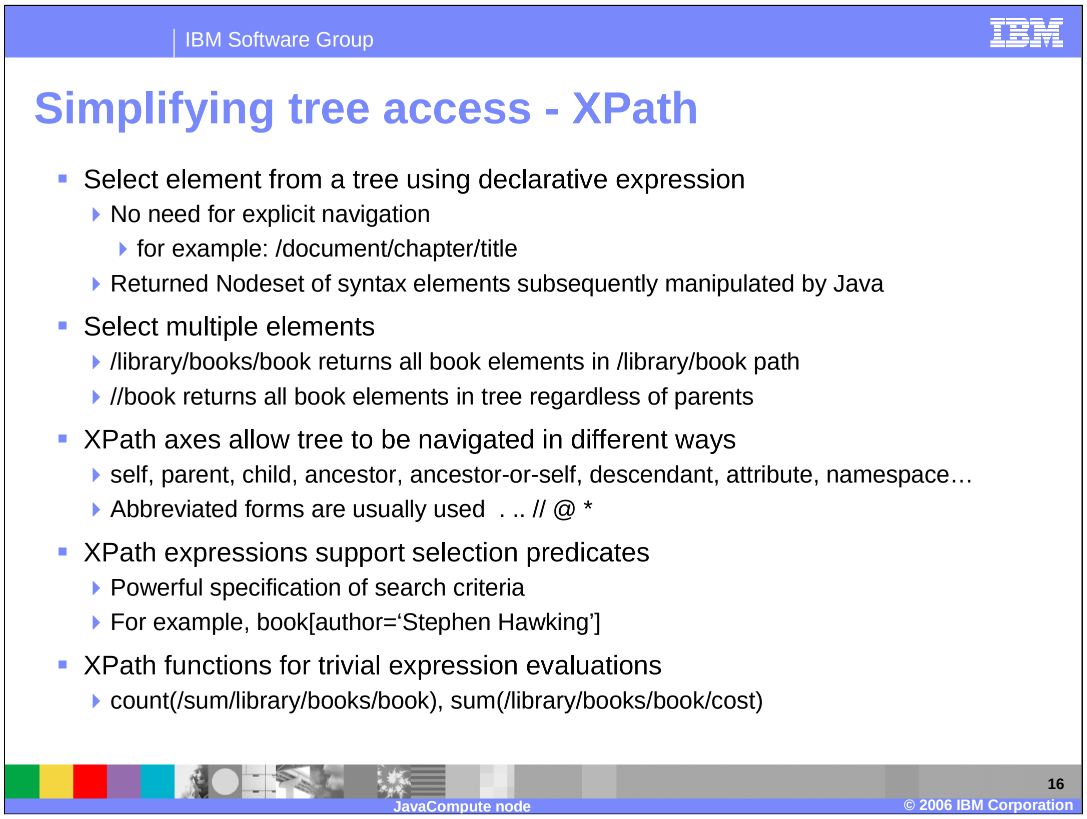
The example on this slide shows a simple XML message, and the logical tree that would be created from the message. The logical tree diagram also shows the methods to call to navigate around the tree.



## Accessing information about an element

- getName()
  - Returns the element name as a java.lang.String
- getValue()
  - Returns the element value
- getType()
  - Returns the generic type, which is one of the following types:
    - NAME. An element of this type has a name, but no value.
    - VALUE. An element of this type has a value, but no name.
    - NAME/VALUE. An element of this type has both a value and a name.
- getSpecificType()
  - Returns the parser-specific type of the element
- getNamespace()
  - Returns the namespace URI of this element

Use the methods listed here to retrieve information about the referenced element from within the JavaCompute node.



## Simplifying tree access - XPath

- Select element from a tree using declarative expression
  - No need for explicit navigation
    - for example: /document/chapter/title
  - Returned Nodeset of syntax elements subsequently manipulated by Java
- Select multiple elements
  - /library/books/book returns all book elements in /library/book path
  - //book returns all book elements in tree regardless of parents
- XPath axes allow tree to be navigated in different ways
  - self, parent, child, ancestor, ancestor-or-self, descendant, attribute, namespace…
  - Abbreviated forms are usually used  . .. // @ *
- XPath expressions support selection predicates
  - Powerful specification of search criteria
  - For example, book[author='Stephen Hawking']
- XPath functions for trivial expression evaluations
  - count(/sum/library/books/book), sum(/library/books/book/cost)

XPath is a query language designed for use with XML documents, but it can be applied to any tree structure for querying purposes. WebSphere Message Broker uses XPath to select elements from the logical message tree regardless of the format of the bit stream.

This slide shows some of the types of XPath statements, which allow a powerful way of accessing data items in a message tree.

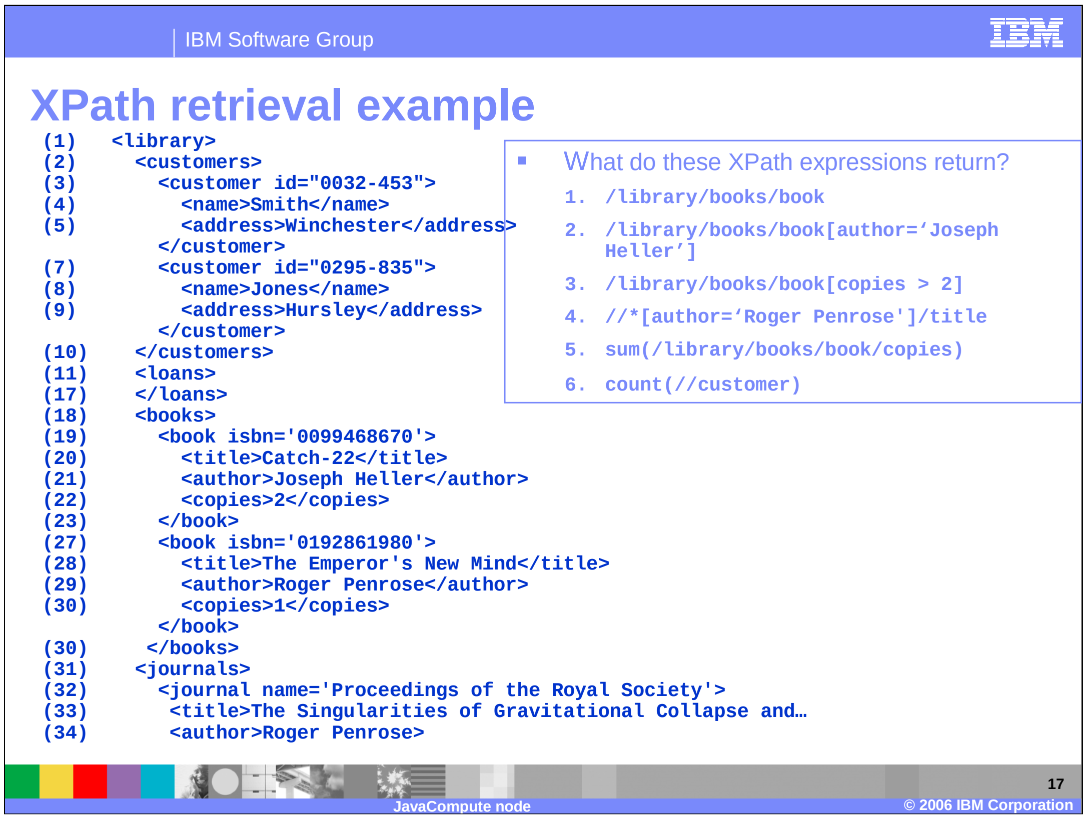
The basic form of XPath is to use a declarative type of expression, such as /document/chapter/title.  This would return the value of the variable “title”.

The second example shows how multiple values can be returned in a single XPath request. This example would return all instances of “book” under the path shown. //book will return all book elements, regardless of where they lie in the tree.

The range of XPath terms include self, parent, child, ancestor, ancestor-or-self, descendant, attribute, namespace, and following-sibling; these are normally abbreviated. A full description of XPath axes is given at www.w3.org/TR/xpath.

XPath supports predicates, so you can search the message tree for any occurrence of a particular value of a variable.

And finally, the XPath syntax provides a number of functions for simple evaluations. For example, it can be used to count the number of occurrences of a variable, or to calculate the sum of a number of variable values which match a set of criteria.


IBM Software Group

IBM

## XPath retrieval example

```
<library>
  <customers>
    <customer id="0032-453">
      <name>Smith</name>
      <address>Winchester</address>
    </customer>
    <customer id="0295-835">
      <name>Jones</name>
      <address>Hursley</address>
    </customer>
  </customers>
  <loans>
  </loans>
  <books>
    <book isbn='0099468670'>
      <title>Catch-22</title>
      <author>Joseph Heller</author>
      <copies>2</copies>
    </book>
    <book isbn='0192861980'>
      <title>The Emperor's New Mind</title>
      <author>Roger Penrose</author>
      <copies>1</copies>
    </book>
   </books>
  <journals>
    <journal name='Proceedings of the Royal Society'>
     <title>The Singularities of Gravitational Collapse and…
     <author>Roger Penrose>
```

- What do these XPath expressions return?

```
1. /library/books/book
2. /library/books/book[author=‘Joseph
   Heller’]
3. /library/books/book[copies > 2]
4. //*[author=‘Roger Penrose']/title
5. sum(/library/books/book/copies)
6. count(//customer)
```

17

JavaCompute node © 2006 IBM Corporation

This slide shows an example of how to use XPath to select parts of the message tree. Although this example uses XML, this function is not restricted to XML.

Example one selects all occurrences of “book” under the tree “library/books”, namely “Catch-22” and “The Emperor’s New Mind’. The returned data would include the attribute of “book”, namely “isbn”. Data lower down the tree would not be returned by this expression, so “title”, “author” and “copies” would not be returned in this example.

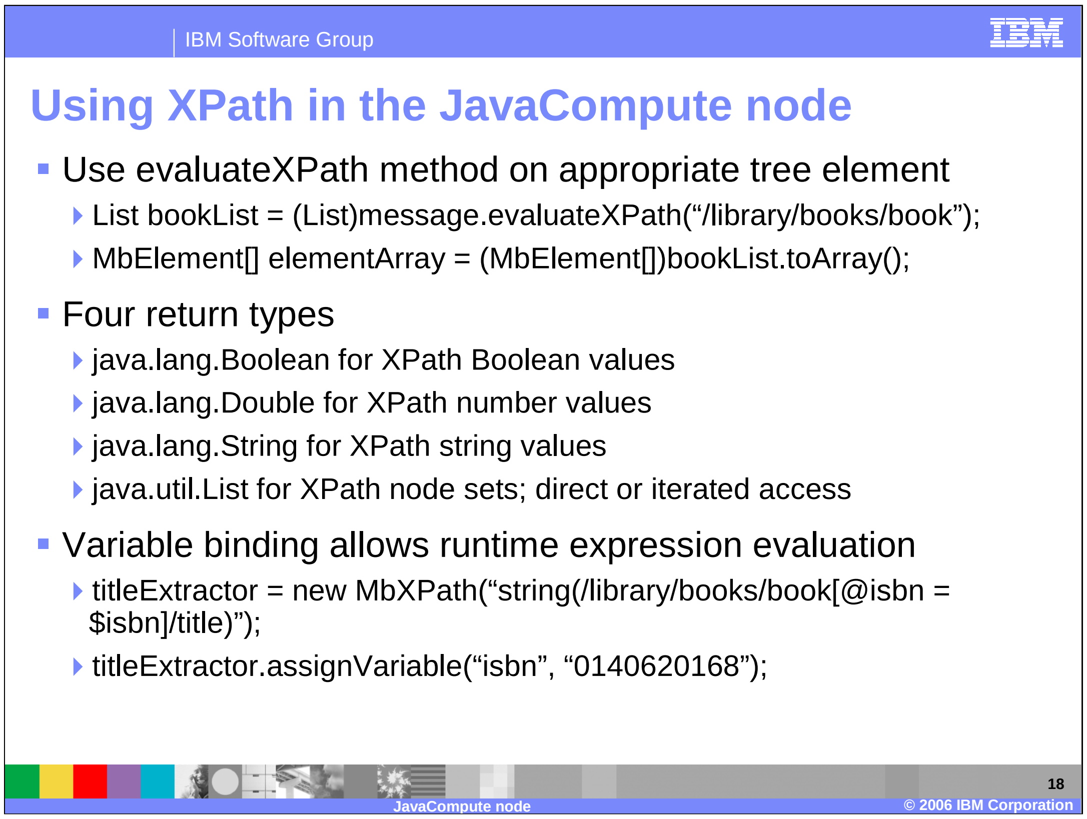
Example 2 uses an expression to further qualify example 1. This will return just one occurrence, namely “Catch-22”. This type of expression can also use the attribute to make a selection. This is denoted by the @ sign in the qualifier.

Example 3 is similar to example 2. In this case, there are no books which satisfy the expression “copies > 2”, so the result is null.

Example 4 uses the double-slash addressing. This means that the expression should look for a result in any part of the tree. This case is simply looking for any match, where “author” has the value “Roger Penrose”. The expression then returns the value “title” corresponding to that match.

Example 5 returns the total number of books fulfilling the expression. In this case the answer is three.

And finally, the count function counts the number of occurrences of the returned matches. The double-slash indicates looking for all occurrences of “customer”;  the answer in this case is two.



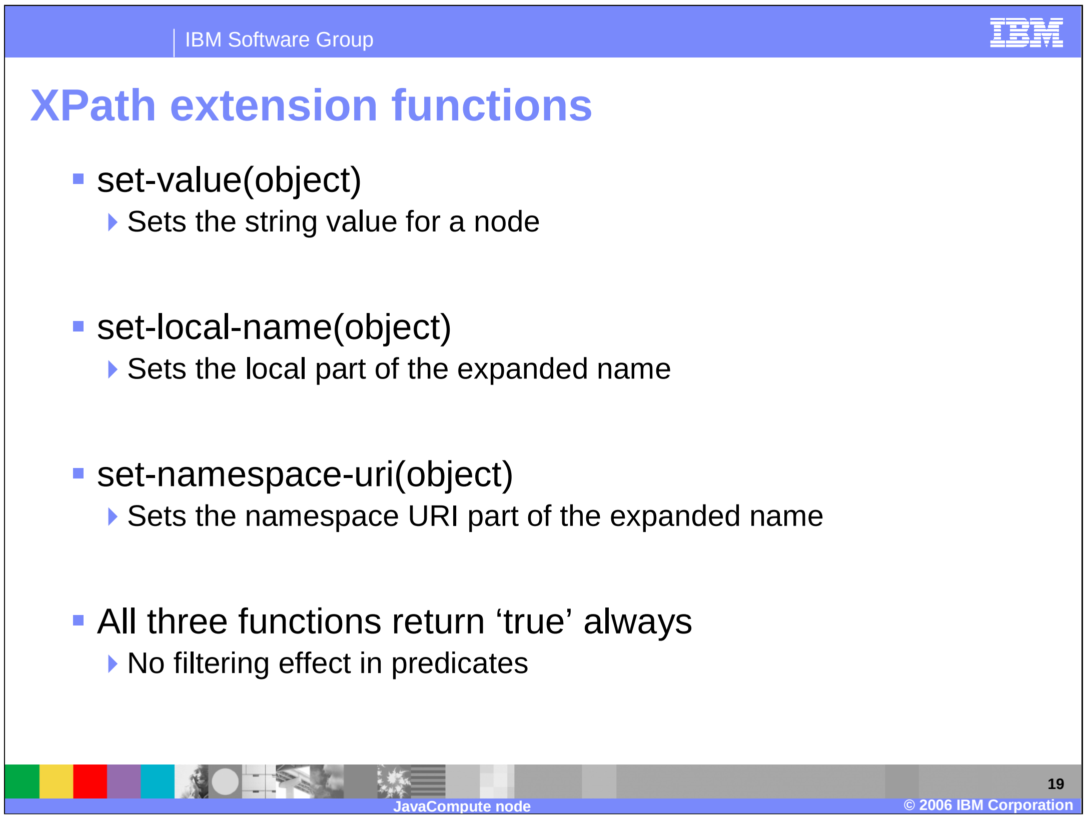
## Using XPath in the JavaCompute node

- Use evaluateXPath method on appropriate tree element
  - List bookList = (List)message.evaluateXPath("/library/books/book");
  - MbElement[] elementArray = (MbElement[])bookList.toArray();
- Four return types
  - java.lang.Boolean for XPath Boolean values
  - java.lang.Double for XPath number values
  - java.lang.String for XPath string values
  - java.util.List for XPath node sets; direct or iterated access
- Variable binding allows runtime expression evaluation
  - titleExtractor = new MbXPath("string(/library/books/book[@isbn = $isbn]/title)");
  - titleExtractor.assignVariable("isbn", "0140620168");

The evaluateXPath() method can be called on a MbMessage object (for absolute paths), or on a MbElement object (for relative paths). The XPath expression is passed to the method as a string parameter. A second form of this method is provided that takes an MbXPath object. This object encapsulates an XPath expression along with variable bindings and namespace mappings, if these are required.

The evaluateXPath() method returns an object of one of these four types, depending on the expression return type, as listed here. The List interface represents an ordered sequence of objects, in this case MbElements. It allows direct access to the elements, or the ability to get an Iterator or an MbElement array.

When using XPath expressions, you can dynamically set values to the variables that are used in the XPath expression. The example shown here assigns a numeric value to the variable "isbn", before using this to obtain the title of the corresponding book.


IBM Software Group

## XPath extension functions

- set-value(object)
  - Sets the string value for a node
- set-local-name(object)
  - Sets the local part of the expanded name
- set-namespace-uri(object)
  - Sets the namespace URI part of the expanded name
- All three functions return ‘true’ always
  - No filtering effect in predicates

19

JavaCompute node

© 2006 IBM Corporation

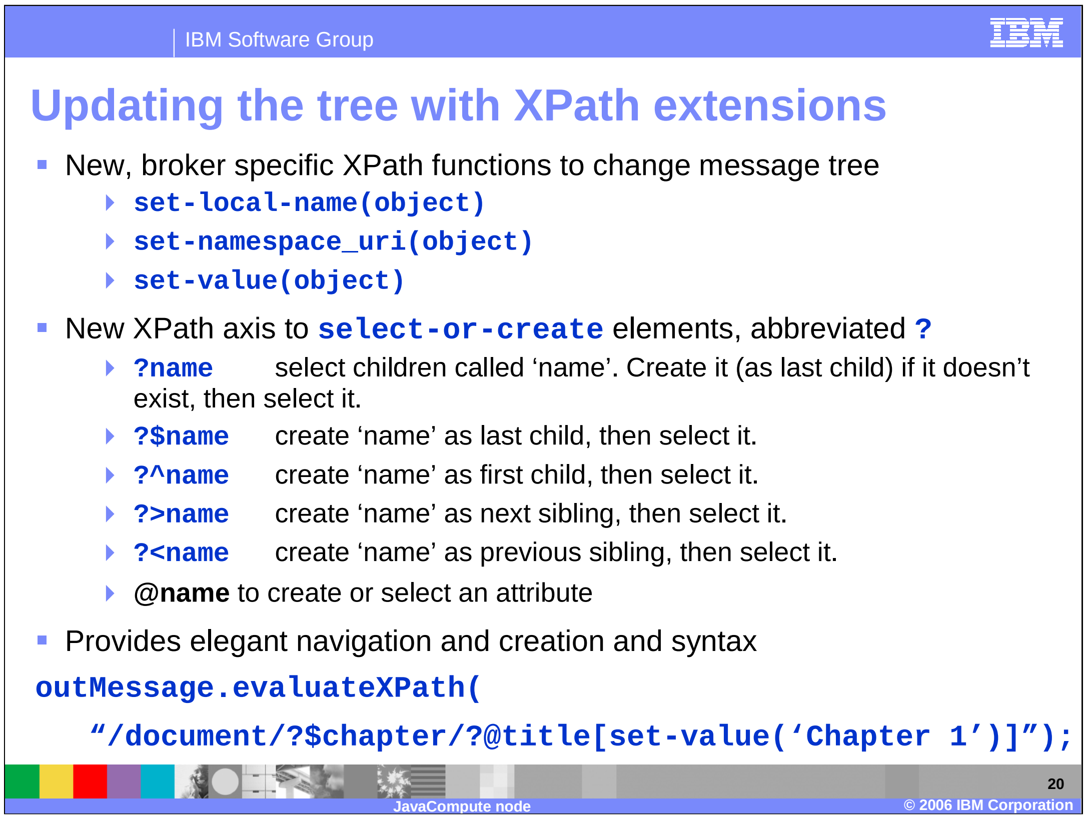
The standard XPath capability does not allow you to create messages. However, in message broker processing, it is important to be able to create or update message trees using this technique.

The WebSphere Message Broker implementation of XPath provides these three extra functions for modifying the message tree.

**set-value** sets the string value of the context node to the value specified in the argument. *object* can be any valid expression and is converted to a string as if a call to the string function were used.

**set-local-name** sets the local part of the expanded name of the context node to the value specified in the argument. *object* can be any valid expression and is converted to a string as if a call to the string function were used.

**set-namespace-uri** sets the namespace URI part of the expanded name of the context node to the value specified in the argument. *object* can be any valid expression and is converted to a string as if a call to the string function were used.


IBM Software Group

## Updating the tree with XPath extensions

- New, broker specific XPath functions to change message tree
  - `set-local-name(object)`
  - `set-namespace_uri(object)`
  - `set-value(object)`
- New XPath axis to `select-or-create` elements, abbreviated `?`
  - `?name` select children called 'name'. Create it (as last child) if it doesn't exist, then select it.
  - `?$name` create 'name' as last child, then select it.
  - `?^name` create 'name' as first child, then select it.
  - `?>name` create 'name' as next sibling, then select it.
  - `?<name` create 'name' as previous sibling, then select it.
  - **@name** to create or select an attribute
- Provides elegant navigation and creation and syntax

```
outMessage.evaluateXPath(
   "/document/?$chapter/?@title[set-value('Chapter 1')]");
```

20

JavaCompute node © 2006 IBM Corporation

The use of this is best illustrated by means of an example, shown on the next slide.

To allow for syntax element trees to be built as well as modified, the following axis is available in addition to the thirteen that are defined in the XPath 1.0 specification:

select-or-create::*name* or ?*name*

?*name* is equivalent to select-or-create::*name*. If *name* is @name, an attribute is created or selected. This selects child nodes matching the specified nametest or creates new nodes according to the following rules:

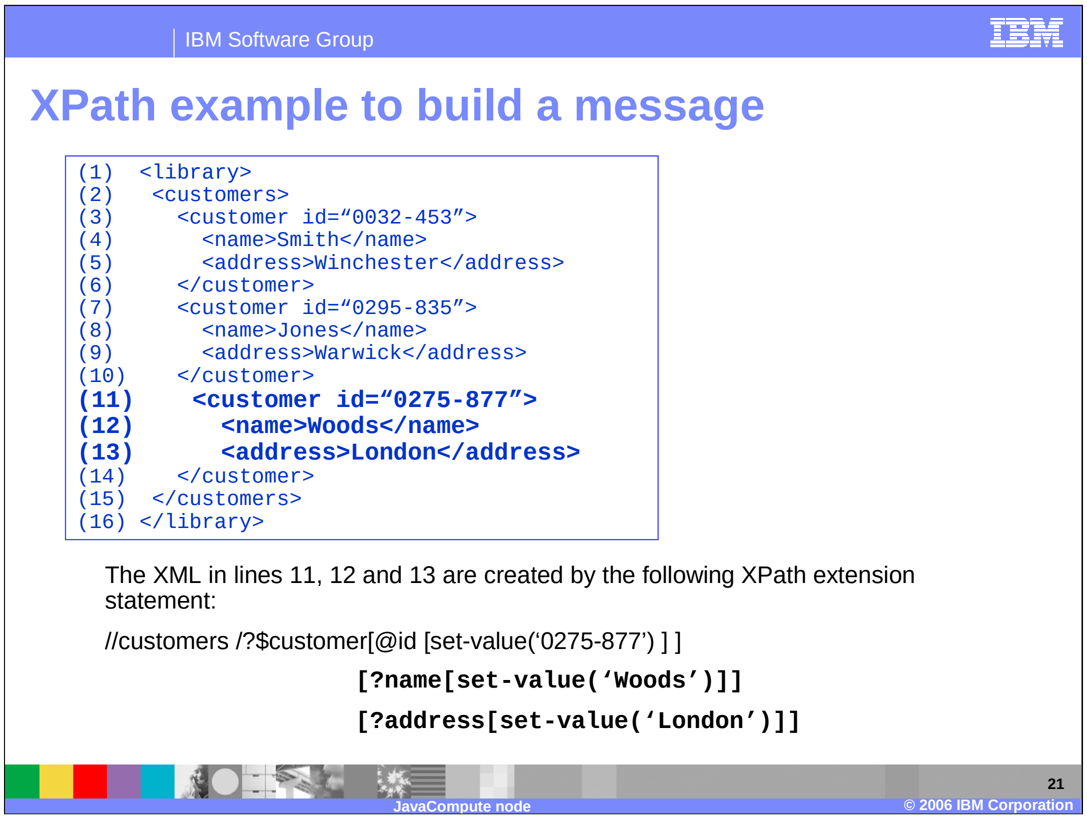
?*name* selects children called *name* if they exist. If there is not a child called *name*, ?*name* creates it as the last child then selects it.

?$*name* creates *name* as the last child, then selects it.

?^*name* creates *name* as the first child, then selects it.

?>*name* creates *name* as the next sibling, then selects it.

?<*name* creates *name* as the previous sibling, then selects it.


IBM Software Group

## XPath example to build a message

```
(1)  <library>
(2)   <customers>
(3)     <customer id="0032-453">
(4)       <name>Smith</name>
(5)       <address>Winchester</address>
(6)     </customer>
(7)     <customer id="0295-835">
(8)       <name>Jones</name>
(9)       <address>Warwick</address>
(10)    </customer>
(11)    <customer id="0275-877">
(12)      <name>Woods</name>
(13)      <address>London</address>
(14)    </customer>
(15)  </customers>
(16) </library>
```

The XML in lines 11, 12 and 13 are created by the following XPath extension statement:

//customers /?$customer[@id [set-value('0275-877') ] ]

```
                    [?name[set-value('Woods')]]
                    [?address[set-value('London')]]
```

21

JavaCompute node

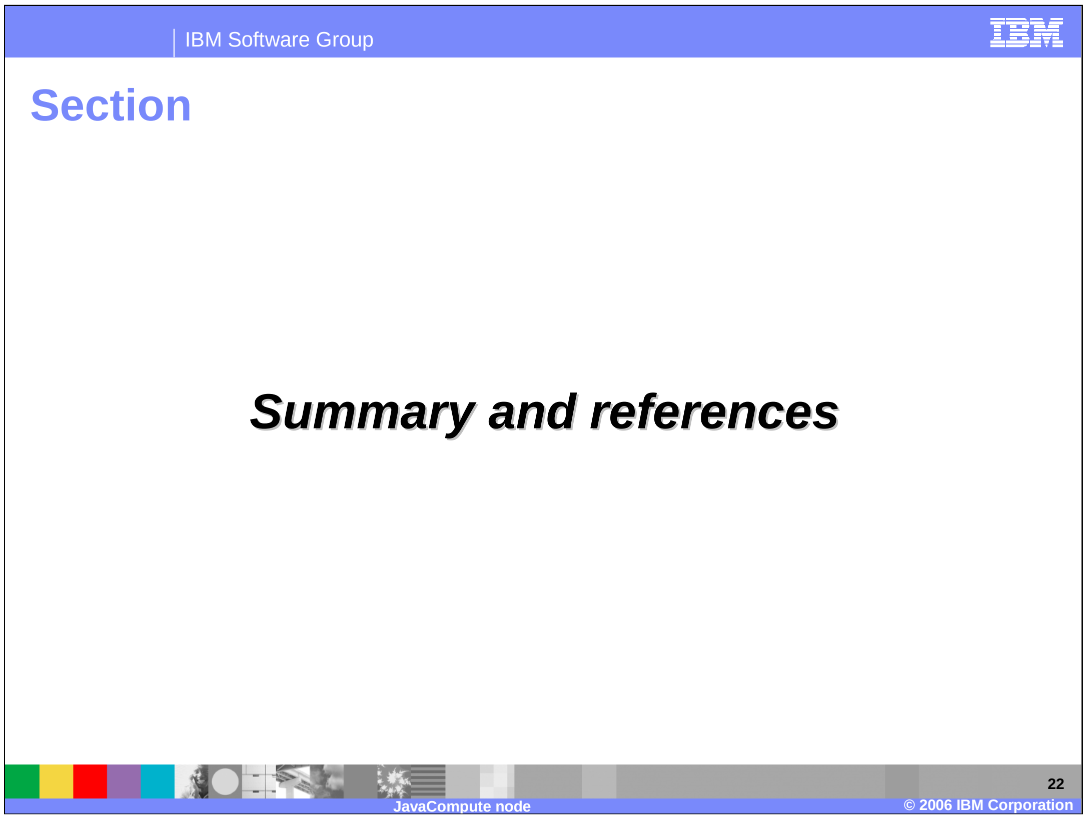
© 2006 IBM Corporation

This example starts with two customers in the XML message tree. To add a third customer with the data values shown in lines 11, 12 and 13 select all elements called "customers", by using the double-slash.

The question mark creates a new element, called customer.  The $ (dollar sign) indicates that it should be created as the last child under "customers".

A new attribute (@) called "id" is created with a value of "0275-877".

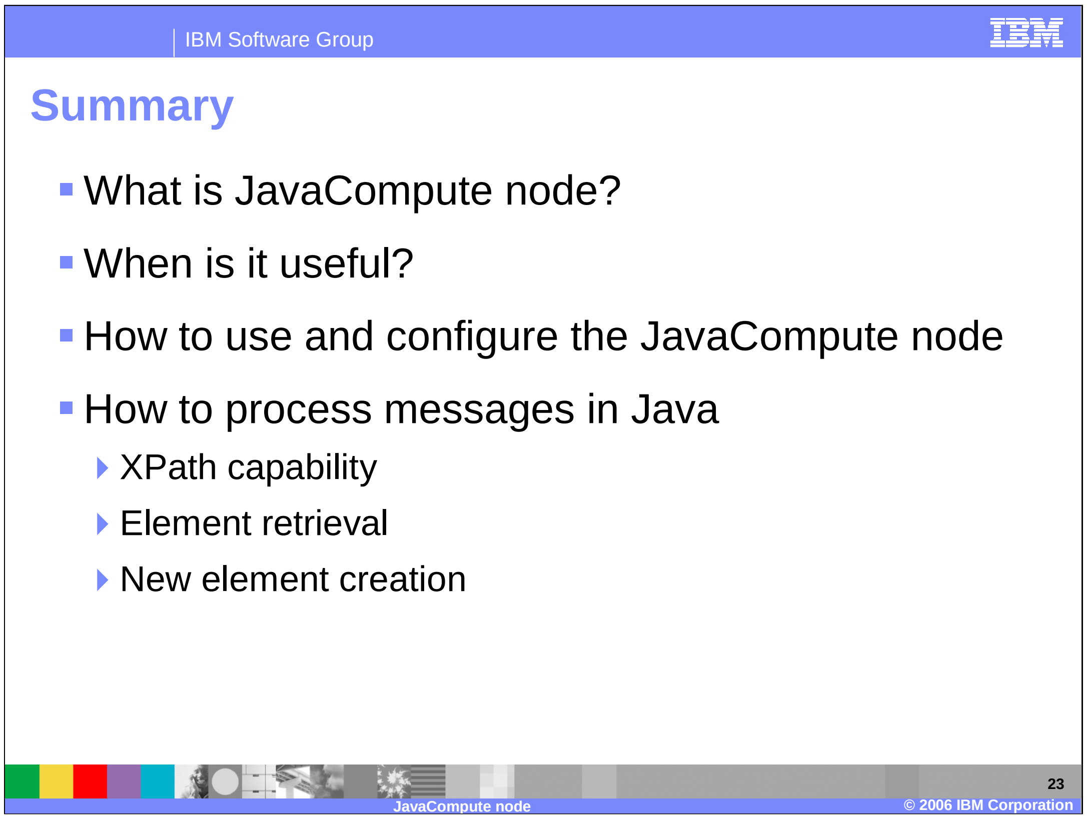
New elements "name" and "address" are created (?) and assigned values.



## Section

***Summary and references***

This section contains a summary and references.


IBM Software Group

## Summary

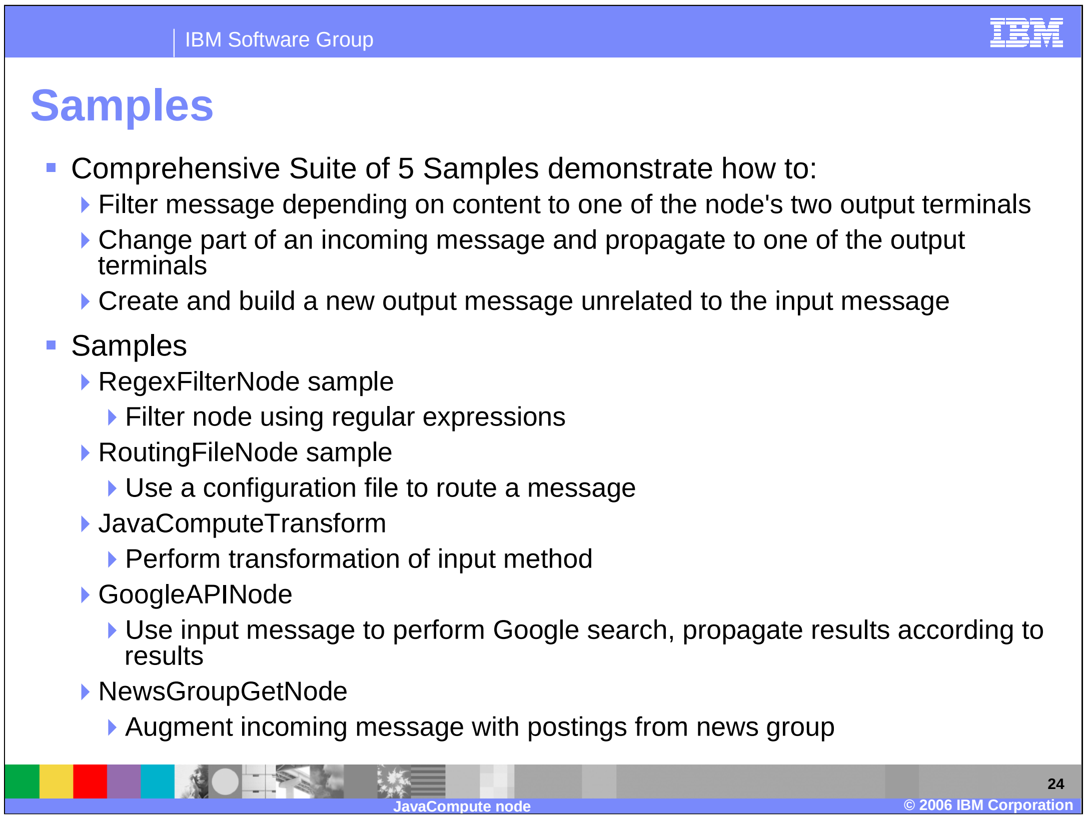
- What is JavaCompute node?
- When is it useful?
- How to use and configure the JavaCompute node
- How to process messages in Java
  - XPath capability
  - Element retrieval
  - New element creation

23

JavaCompute node

© 2006 IBM Corporation

To summarize, this session provided an introduction to the JavaCompute node, and explained when and how to use this function. It described how to use it to process messages, how to use it in conjunction with XPath for data transformation, and how to access individual elements of the message tree.

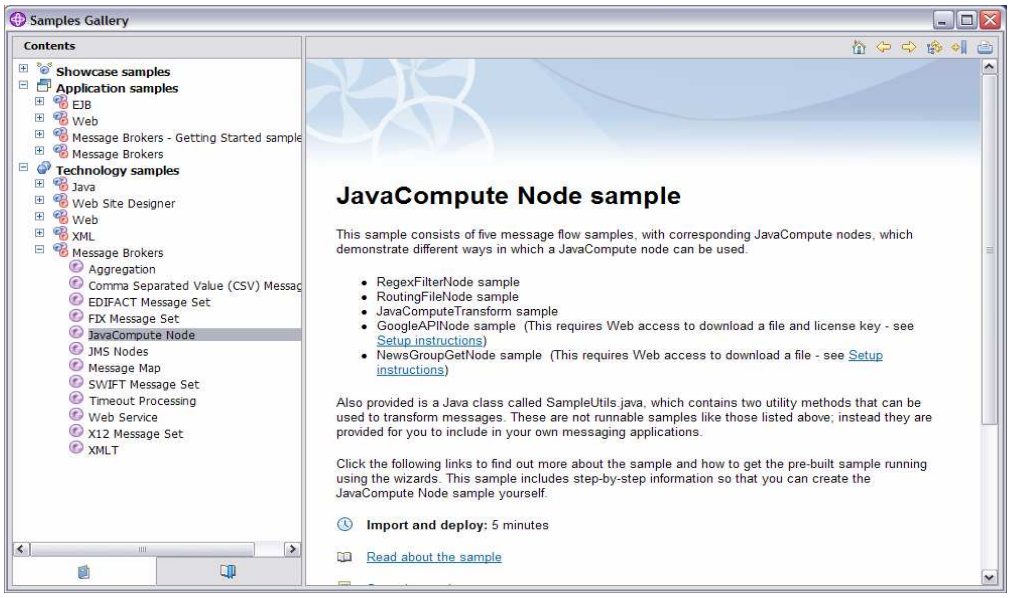


## Samples

- Comprehensive Suite of 5 Samples demonstrate how to:
  - Filter message depending on content to one of the node's two output terminals
  - Change part of an incoming message and propagate to one of the output terminals
  - Create and build a new output message unrelated to the input message
- Samples
  - RegexFilterNode sample
    - Filter node using regular expressions
  - RoutingFileNode sample
    - Use a configuration file to route a message
  - JavaComputeTransform
    - Perform transformation of input method
  - GoogleAPINode
    - Use input message to perform Google search, propagate results according to results
  - NewsGroupGetNode
    - Augment incoming message with postings from news group

The Samples Gallery provides a number of samples illustrating how Java can be used to do different functions. These cover simple examples such as transformation and routing, as well as more complex usage covering the invocation of external services such as a search engine or a News group feed.

## Samples Gallery



JavaCompute Node sample

This sample consists of five message flow samples, with corresponding JavaCompute nodes, which demonstrate different ways in which a JavaCompute node can be used.

- RegexFilterNode sample
- RoutingFileNode sample
- JavaComputeTransform sample
- GoogleAPINode sample (This requires Web access to download a file and license key - see Setup instructions)
- NewsGroupGetNode sample (This requires Web access to download a file - see Setup instructions)

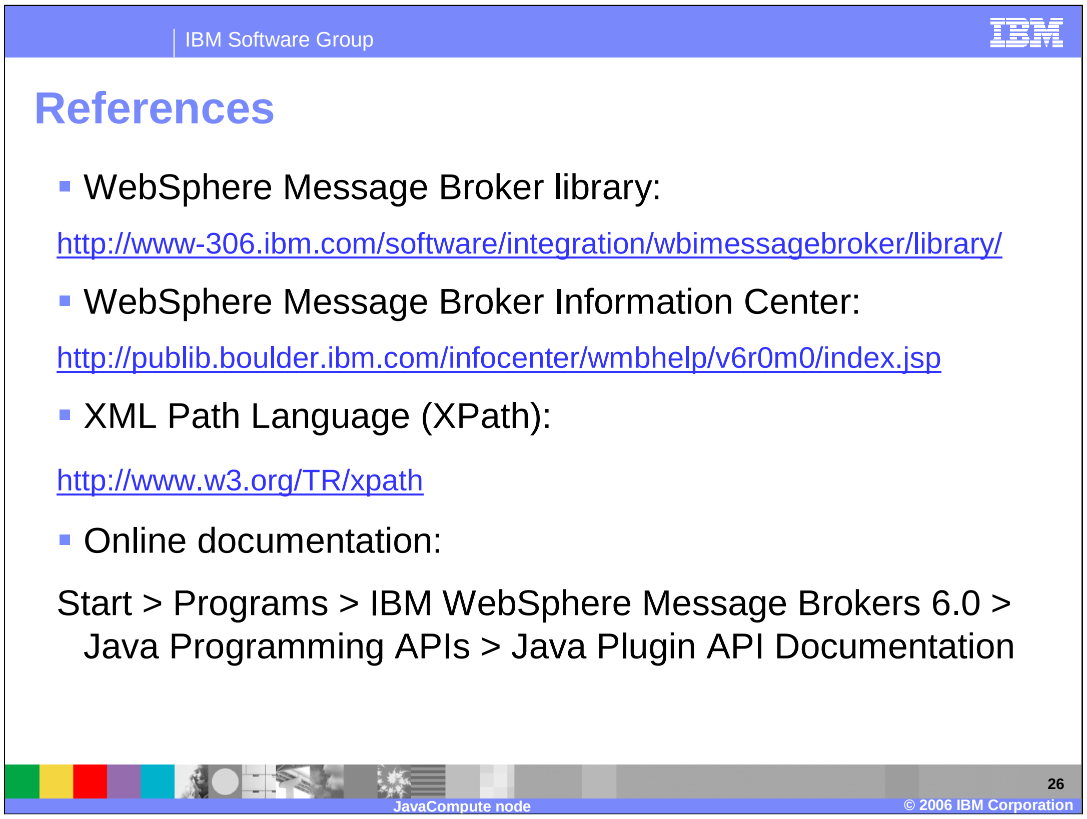
Also provided is a Java class called SampleUtils.java, which contains two utility methods that can be used to transform messages. These are not runnable samples like those listed above; instead they are provided for you to include in your own messaging applications.

Click the following links to find out more about the sample and how to get the pre-built sample running using the wizards. This sample includes step-by-step information so that you can create the JavaCompute Node sample yourself.

**Import and deploy:** 5 minutes

Read about the sample

The Samples Gallery can be accessed from your toolkit by selecting Help, Samples Gallery.



## References

- WebSphere Message Broker library:

http://www-306.ibm.com/software/integration/wbimessagebroker/library/

- WebSphere Message Broker Information Center:

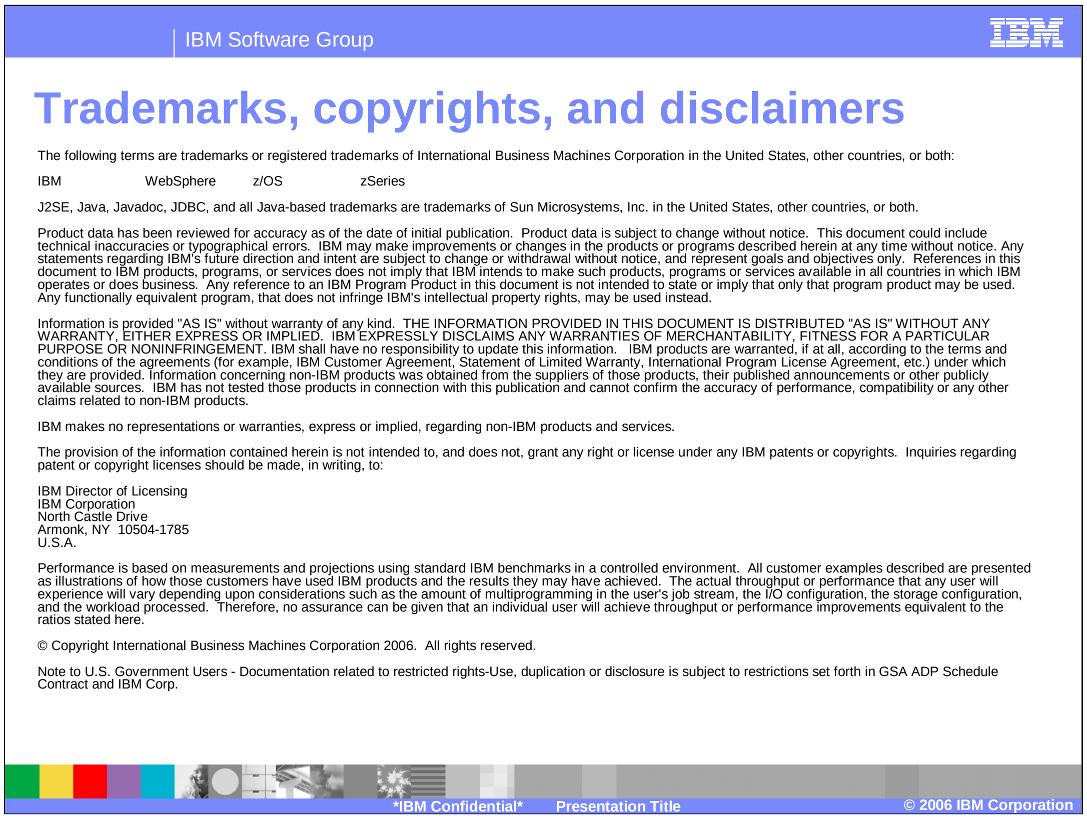
http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp

- XML Path Language (XPath):

http://www.w3.org/TR/xpath

- Online documentation:

Start > Programs > IBM WebSphere Message Brokers 6.0 > Java Programming APIs > Java Plugin API Documentation



## Trademarks, copyrights, and disclaimers