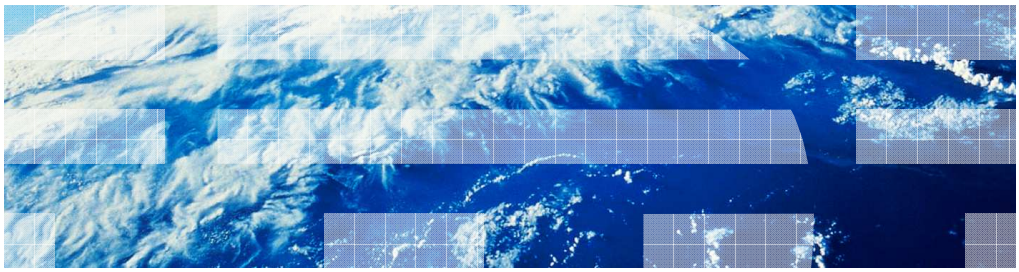
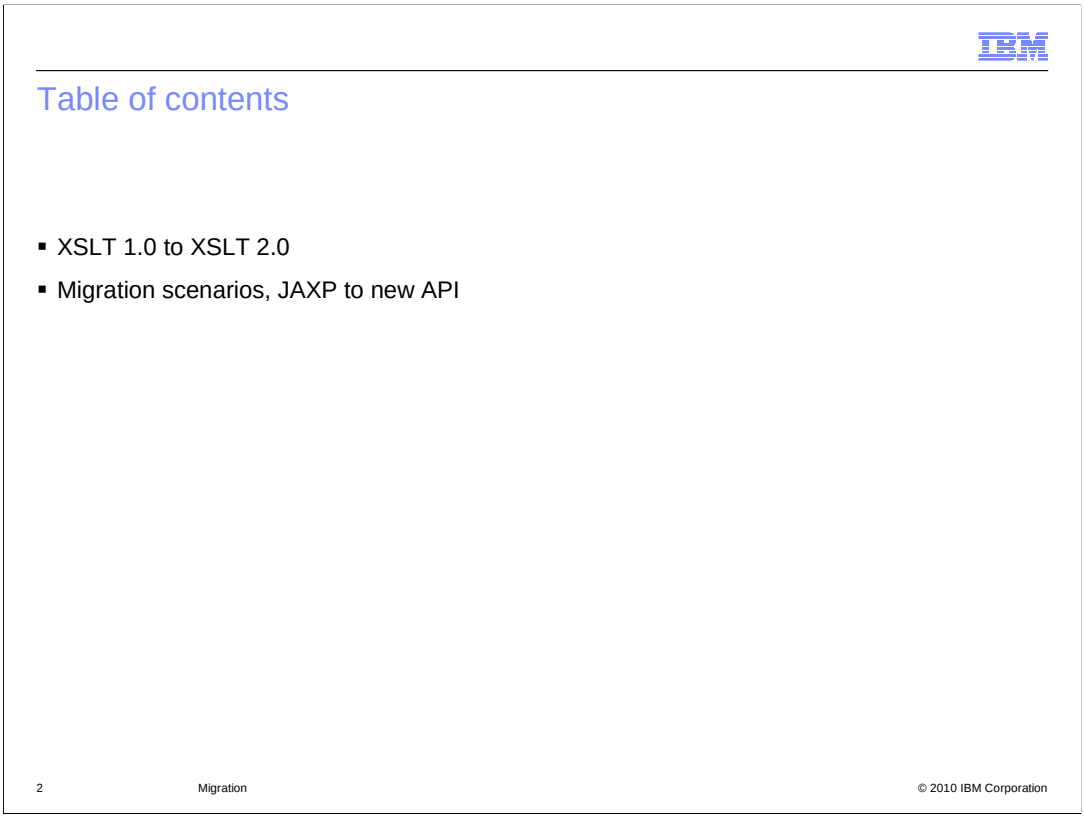
IBM WebSphere Application Server Feature Pack for XML

Migration

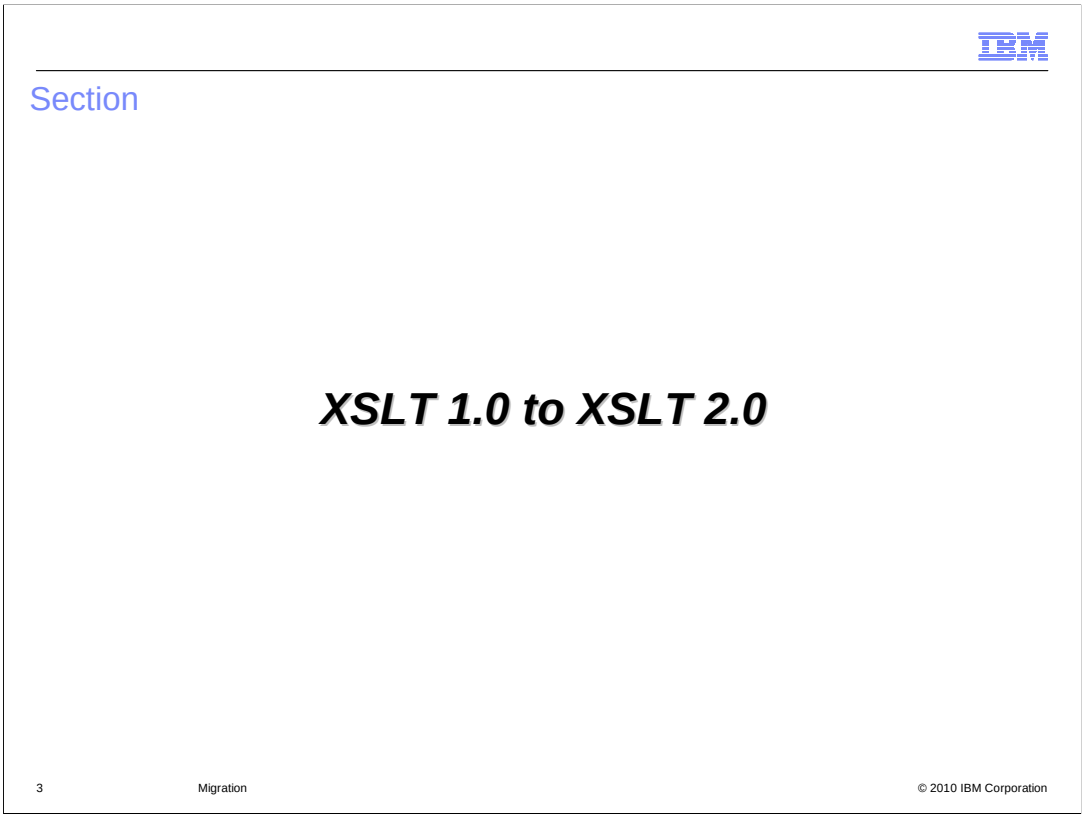


WebSphere software

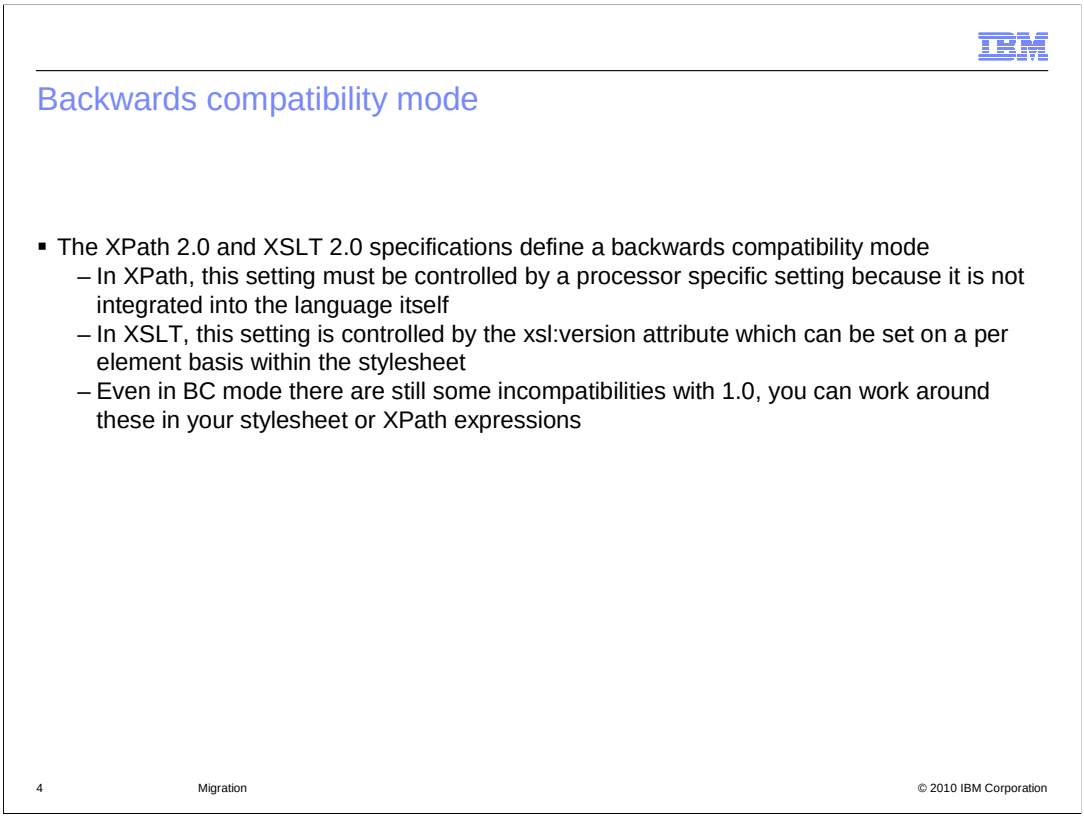
This presentation will cover the migration scenarios of going from XSLT 1.0 to XSLT 2.0 and the migration scenarios from JAXP to the new XML API.



Listed here is the agenda for this presentation.



First, this presentation will walk through the migration from XSLT 1.0 to XSLT 2.0.



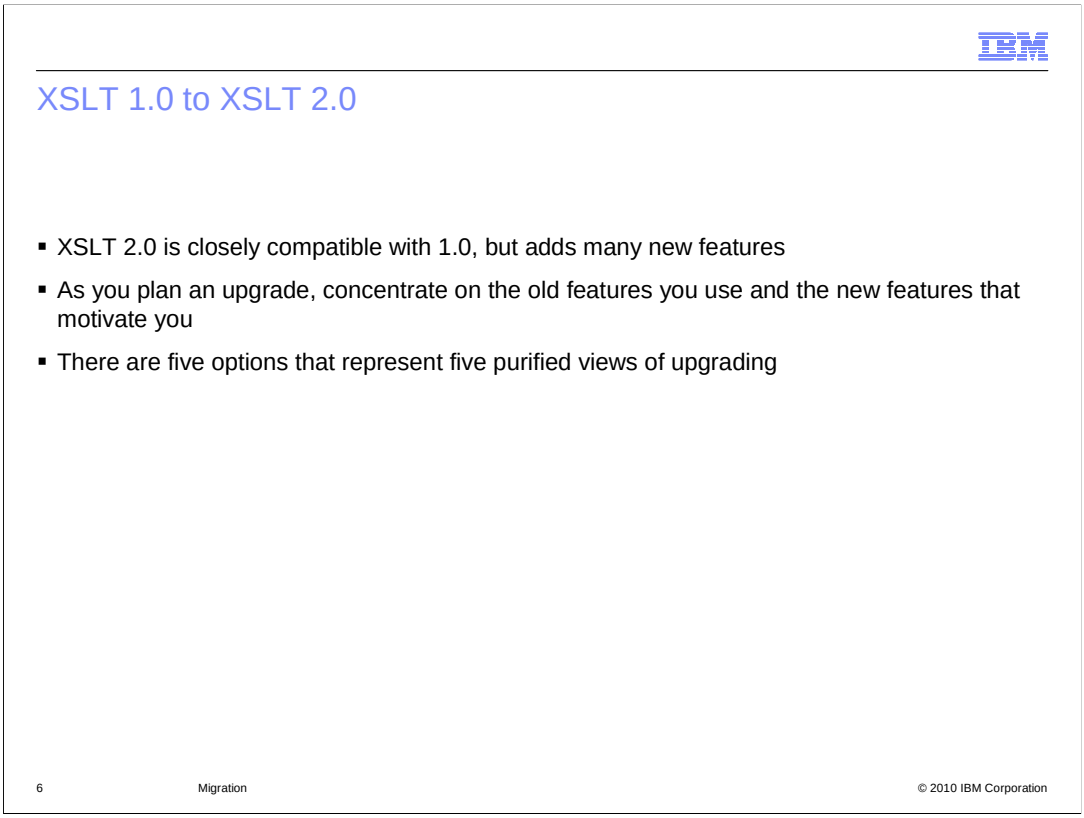
It's great to have new functions and features, but what about existing XPath 1.0 and XSLT 1.0 applications?

The XPath 2.0 and XSLT 2.0 specifications define a backwards compatibility mode that minimizes the differences between pure 1.0 behavior and 1.0 backwards compatible behavior. In XPath, this setting must be controlled by a processor specific setting because it is not integrated into the language itself. In XSLT, this setting is controlled by the xsl:version attribute which can be set on a per element basis within the stylesheet. The xsl:use-when attribute can be used to have different behavior on different versions. Even in BC mode there are still some incompatibilities with 1.0, you can work around these in your stylesheet or XPath expressions.

Applications

- Suggestions for migrating applications from older standards to newer standards
  - Change from running on 1.0 processors to running in 1.0 backwards compatible mode on 2.0 processors
  - Look for opportunities to simplify your expressions and stylesheets in moving to 2.0 over time
- Suggestions for new application - start with the newer 2.0 standards

For your applications the suggestion for migrating applications from older standards to newer standards is to either change from running on 1.0 processors to running in 1.0 backwards compatible mode on 2.0 processors or look for opportunities to simplify your expressions and stylesheets in moving to 2.0 over time. For a new application, go ahead and start with the newer 2.0 standards.



You might have heard that XSLT 2.0 is closely compatible with 1.0 and that it adds many new features. Your 1.0 stylesheets probably uses only a fraction of the 1.0 features and uses a smaller fraction of the 2.0 feature set. As you plan an upgrade, concentrate on the old features *you* use and the new features that motivate *you*. This presentation follows five options that represent views of upgrading. You will want to consider two kinds of decision factors when you choose an option: organizational capability factors and impact factors deriving from the 2.0 features that appeal to you.

Option 1: Full rewrite

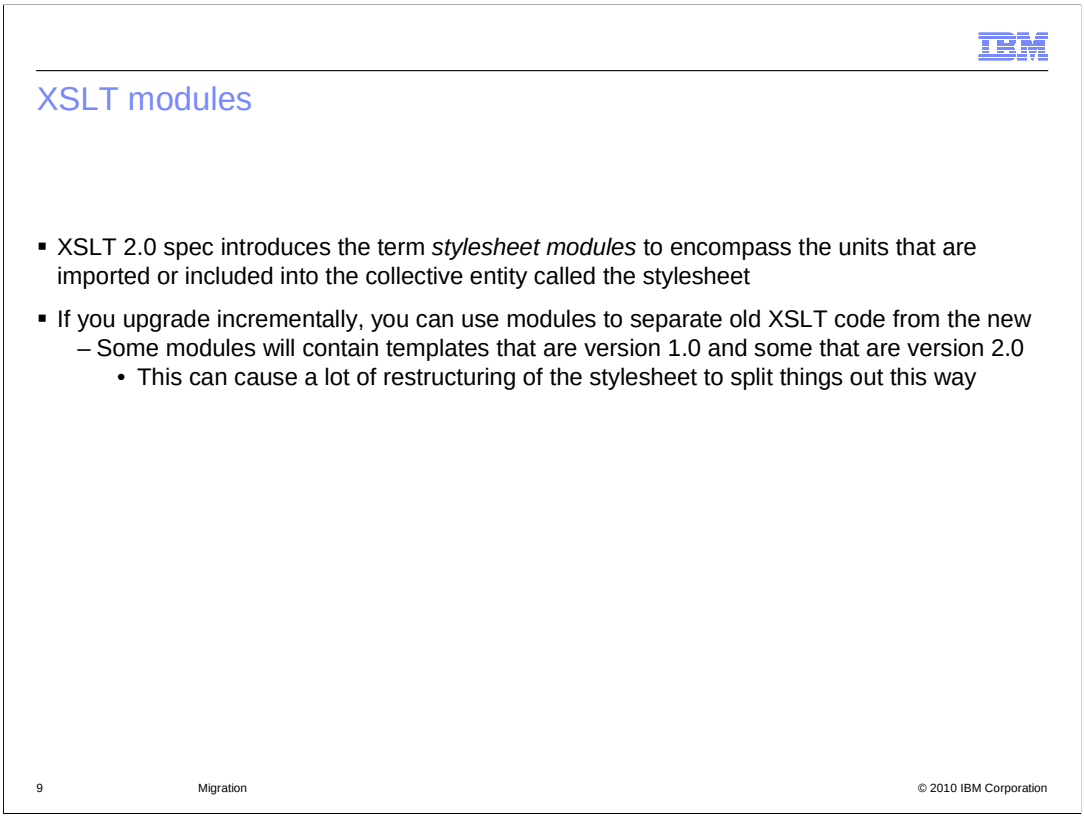
- You can take full advantage of 2.0 syntax and features because you are basically starting from scratch
- You can make your code more readable by using 2.0 features
- You can make your code more straightforward because it does not involve backwards compatibility
- The result is portable across 2.0 processors

Option one is a full re-write. If you choose this option, then it's like starting from the beginning. To take full advantage of XSLT 2.0 syntax and features, it is necessary to plan, possibly change the whole design, and probably rewrite most of your stylesheet modules. If you have a good modular structure with your current stylesheets, you might choose to retain the structure. More generally, if you have a clean architecture for 1.0 already in place, you might save some planning time, depending on which 2.0 features you want to exploit.

Option 2: Convert most to 2.0

- This option is the best approach for a gradual transition, if you eventually want to eliminate backwards compatibility
- You can enhance modularity and reusability of your code
- You can apply locally standard tweaks or code cleanup as you go
- The result is vulnerable to mistakes in local version handling and tweaks
- It might consume more coding time on branching code for alternative versions

If a complete rewrite to 2.0 seems too drastic, then the next best option is to convert most of your stylesheet modules to 2.0 and reuse some 1.0 modules. You can make the 1.0 islands at the level of a single template or even deeper. This option still requires some planning, particularly about the places where you will use new 2.0 features, and investigating the reusability of old modules in your new stylesheet structures.



XSLT 2.0 specification introduces the term *stylesheet modules* to encompass the units that are imported or included into the collective entity called the stylesheet. If you upgrade incrementally, you can use modules to separate old XSLT code from the new. Some modules will contain templates that are version 1.0 and some that are version 2.0. This can cause a lot of restructuring of the stylesheet to split things out this way.

Option 3: Rewrite modules

- You can learn new features incrementally
- You can focus on performance bottlenecks
- You can apply locally standard tweaks or code cleanup as you go
- This is a patchy approach that might cause errors due to the wrong version being in effect
- It is harder to capitalize on new features that have wide impact, such as schema-awareness
- Requires knowing both versions to debug

If specific 2.0 features are desirable, then you can try to patch 2.0 version islands in your 1.0 stylesheet, and run it using a 2.0 processor that supports backwards compatibility. If the processor generates unexpected results, then tweak your 1.0 modules until they work. This option lets you capitalize on the 2.0 features you want with minimal planning or redesign. Naturally, this works best if the new features you want are easily isolated. Some benefits of this approach include that you can learn new features incrementally and focus on performance bottlenecks. However, this can be a patchy approach that might cause errors, also it requires having knowledge of both versions.

Option 4: Change to 2.0 and debug from there

- You can take an incremental approach to conversion
- This might be the fastest conversion if your 1.0 stylesheet did not run afoul of the compatibility exceptions
- You can apply locally standard tweaks or code cleanup as you go
- Without initial planning, it is harder to capitalize on new features that have wide impact, such as schema-awareness
- It is hard to predict how long the conversion will take

If you need to change your existing 1.0 stylesheets into 2.0 stylesheets quickly, then why not just set the stylesheet version attribute to 2.0 and see if you get the expected output from a 2.0 processor? If that does not work, then start your debugging by back-tracking the result of each template and tweak until it works. Tweaking might involve fixing it with the appropriate 2.0 instruction, or (if you have the BC feature) you can just set the version attribute to 1.0 and see what happens. Of course, this option shows much less concern for exploiting new 2.0 features.

Option 5: Stay at 1.0

- No work and no new processor needed
- You do not have to read six new specifications
- 1.0 processors are mature and well debugged.
- You cannot take advantage of new 2.0 features
- Turnaround time for processor bug fixing can be slower because XSLT product developers might not invest as much in 1.0 anymore
- The old techniques in XSLT 1.0 can become more irritating over time when you know that a readable 2.0 replacement exists

You might ask, “why switch at all when your 1.0 stylesheets work as expected?” You do not have to learn 2.0 syntax, do not have to plan for a 2.0 stylesheet structure, and do not have to install a 2.0 processor. This might be a viable option for you, but remember that 2.0 can do some things that a 1.0 processor cannot do, and 2.0 addresses many shortcomings of 1.0.

The next few slides help with the migration from JAXP to the new XML API.

Migration

- Existing JAXP XPath 1.0, XSLT 1.0 applications will continue to run on WebSphere® Application Server V7 XML runtime
- These applications can be easily converted to the new XML runtime by
  - Converting the invocation API to the XML Feature Pack API (ease of conversion dependent on depth of JAXP usage)
  - Setting the backwards compatibility flag in the API (for XPath 1.0) or the version attribute in the stylesheet (for XSLT 1.0)
    - Backwards compatibility, as defined by the specification, handles most incompatibilities between 1.0 and 2.0
- In the long term move these XPath and XSLT applications to the 2.0 version and runtime
  - Take advantage of functional enhancements and reduced complexity

First remember that existing JAXP XPath 1.0, XSLT 1.0 applications will continue to run on WebSphere Application Server V7 XML runtime. Next think about converting to the new XML runtime by converting the invocation API to the XML Feature Pack API. To help, you can set the backwards compatibility flag in the API (for XPath 1.0) or the version attribute in the stylesheet (for XSLT 1.0). Of course, you should in the long term move these XPath and XSLT applications to the 2.0 version and runtime in order to take advantage of functional enhancements and reduced complexity.

Processing an XSLT stylesheet using the API

- Example below shows how to process an XSLT stylesheet and apply it to some input to produce an instance of the javax.xml.transform.Result interface

```
XFactory factory = XFactory.newInstance();
XSLTExecutable style = factory.prepareXSLT(new StreamSource("style.xsl"));
style.execute(new StreamSource("input.xml"), new StreamResult(System.out));
```

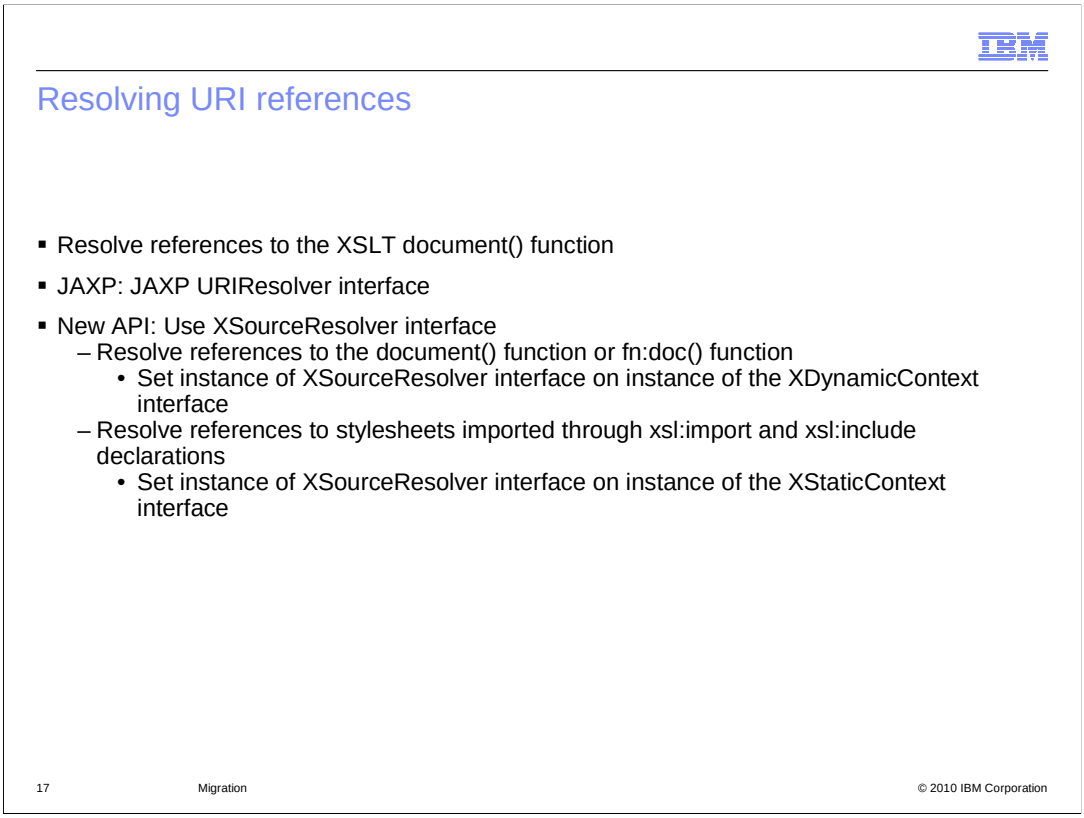

Here is a basic example of how to process an XSLT stylesheet and apply it to some input to produce an instance of the javax.xml.transform.Result interface.

Processing an XPath expression using the API

- The example below demonstrates how to process an XPath expression and apply it to some input

```
XFactory factory = XFactory.newInstance(); XPathExecutable pathExpr =
       factory.prepareXPath("/doc/child[@id='N1378']");

// Process input from a StreamSource

XSequenceCursor result1 = pathExpr.execute(new    StreamSource("input.xml"));

// Process input from a DOM node

XSequenceCursor result2 = pathExpr.execute(new    DOMSource(node));
```


Here is an example that demonstrates how to process an XPath expression and apply it to some input.



If you used instances of the JAXP URIResolver interface to resolve references to the XSLT document() function, you can now use the XSourceResolver interface to accomplish the same thing. To resolve references to the document() function or the fn:doc() function, you can set an instance of the XSourceResolver interface on an instance of the XDynamicContext interface. To resolve references to stylesheets imported through xsl:import and xsl:include declarations, you can set an instance of the XSourceResolver interface on an instance of the XStaticContext interface.

Extension functions and external functions

- Supply the implementations of extension functions that your XPath expressions might call
- JAXP: register an instance of XPathFunctionResolver interface
  - XPath expressions can call extension functions you register
- New API:
  - declare extension functions on an instance of the XStaticContext interface
    - Specify the expected types of the arguments and the expected type of the result of calling the function
  - Register the implementations of extension functions on an instance of the XDynamicContext interface
    - XSLT stylesheet and XPath and XQuery expressions can call any extension functions that you register

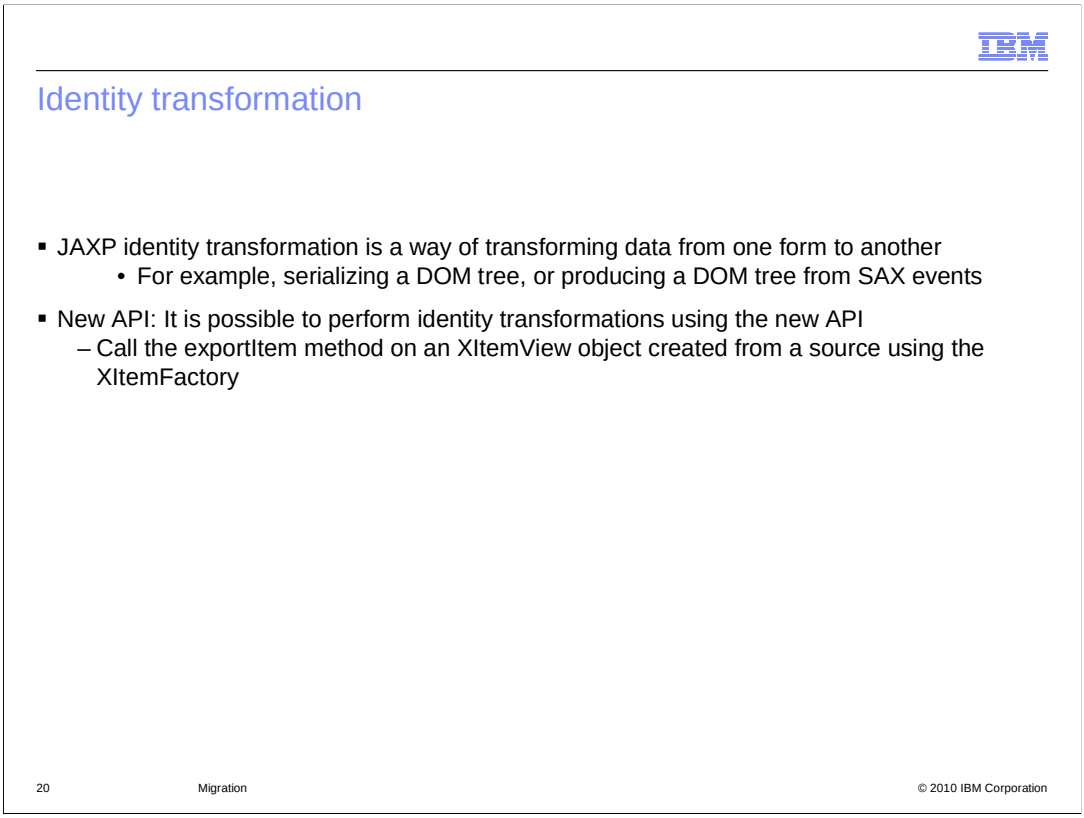
Using JAXP for XPath expression evaluation, you can register an instance of the XPathFunctionResolver interface to supply the implementations of extension functions that your XPath expressions might call. The XSLT portion of JAXP does not have an equivalent mechanism.

With the current API, you can declare extension functions on an instance of the XStaticContext interface. You will specify the expected types of the arguments and the expected type of the result of calling the function, and you can register the implementations of extension functions on an instance of the XDynamicContext interface. Your XSLT stylesheet and XPath and XQuery expressions can call any extension functions that you register.

Stylesheet parameters and external variables

- JAXP
  - Initial values of stylesheet parameters: Transformer.setParameter method
  - Values of variables for XPath expressions: instance of the XPathVariableResolver interface
- New API:
  - Declare variables: declareVariable() methods of the XStaticContext interface
    - For XSLT and XQuery, variables must be declared in the stylesheet or query itself and should not be re-declared in the static context. Only for XPath do variables need to be declared in the static context.
  - Supply the values of stylesheet parameters, XPath variables, and XQuery external variables through bind() methods of the XDynamicContext interface

Using JAXP, you can supply the initial values of stylesheet parameters by calling the Transformer.setParameter method and you can supply the values of variables for XPath expressions by supplying an instance of the XPathVariableResolver interface. Using the API, you can declare variables using the declareVariable() methods of the XStaticContext interface, specifying a variable name and the expected type of the variable. You can supply the values of stylesheet parameters, XPath variables, and XQuery external variables through one of the bind() methods of the XDynamicContext interface.



An operation that is frequently used in JAXP is the identity transformation. This is a way of transforming data from one form to another for example, serializing a DOM tree, or producing a DOM tree from SAX events. It is possible to perform identity transformations using the new API. You need to call the exportItem method on an XItemView object created from a source using the XItemFactory. An identity XSLTExecutable cannot be created by preparing a null stylesheet.

JAXP: Prepare-time and execution-time

- In JAXP runtime configuration information for XSLT stylesheets is supplied directly on
  - the objects that are used to perform transformations
  - instances of the Transformer interface
  - TransformerHandler interface
- Similarly, you supply configuration information for the preparation of stylesheets and XPath expressions directly on instances of
  - TransformerFactory classes
  - XPathFactory classes
- Instances of the Transformer, TransformerHandler and XPathExpression interfaces are not thread safe

In JAXP you supply much of the runtime configuration information for XSLT stylesheets such as the values of stylesheet parameters, URIResolvers, and so on. You supply this information directly on the objects that are used to perform transformations – instances of the Transformer interface and the TransformerHandler interface. Similarly, you supply configuration information for the preparation of stylesheets and XPath expressions directly on instances of the TransformerFactory and XPathFactory classes in JAXP. Every thread that uses them has to synchronize access to shared instances of those objects or create distinct copies that are specific to each thread.

API: Prepare-time and execution-time

- New API, supply configuration information that is needed at the time a stylesheet or an expression is prepared
  - Use instance of the XStaticContext interface
  - Pass as an argument to the prepare methods of the XFactory class
- Similarly, provide any configuration information that is needed to evaluate a stylesheet or expression
  - Use instance of the XDynamicContext interface
  - Pass as an argument to the execute methods of the XExecutable interface and its subinterfaces
- This separation of the configuration information into a separate object makes the API more thread safe

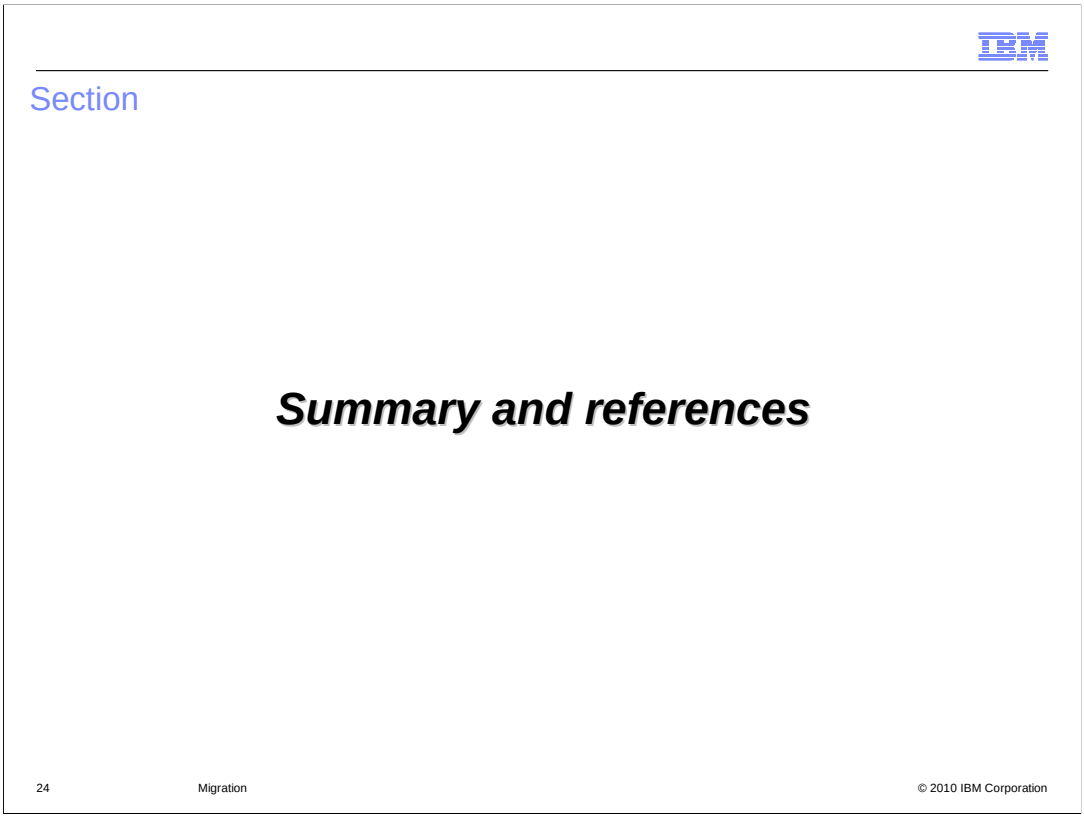
With the new API, you can supply configuration information that is needed at the time a stylesheet or an expression is prepared such as namespace bindings, the types of external functions or variables, and so on. This information is supplied using an instance of the XStaticContext interface. Similarly, you can provide any configuration information that is needed to evaluate a stylesheet or expression such as the values of variables, settings of output parameters, and so on. This information is supplied on an instance of the XDynamicContext interface, which you can pass as an argument to the execute methods of the XExecutable interface and its subinterfaces.

This separation of the configuration information into a separate object makes the API more thread safe. Your application can use the same instance of the XExecutable interface on different threads without any synchronization. This stands in contrast to JAXP, where instances of the Transformer, TransformerHandler and XPathExpression interfaces are not thread safe. In JAXP, every thread that uses them has to synchronize access to shared instances of those objects or create distinct copies that are specific to each thread.

Handling errors

- JAXP, you can supply an instance of the ErrorHandler interface to control how the processor should respond to errors
- New API, supply an instance of the XMessageHandler interface on an instance of
  - XStaticContext interface for preparation-time errors or
  - XDynamicContext interface for execution-time errors

In JAXP, you can supply an instance of the ErrorHandler interface to control how the processor should respond to errors. In the API, you can achieve this by supplying an instance of the XMessageHandler interface on an instance of the XStaticContext interface for preparation-time errors or the XDynamicContext interface for execution-time errors.



The next section provides a summary and references

Summary

- XSLT 1.0 to XSLT 2.0
- 5 choices to take for migration
- Migration scenarios, JAXP to new API

This presentation went through the migration scenarios migrating from XSLT 1.0 to XSLT 2.0. It also went through the migration scenarios to go from JAXP to the new XML API.

References (1 of 2)

- WebSphere Application Server Feature Pack for XML
  - http://www.ibm.com/software/webservers/appserv/was/featurepacks/xml/
- Infocenter
  - http://www14.software.ibm.com/webapp/wsbroker/redirect?version=v700xml&product=was-nd-mp
- Primary specifications
  - http://www.w3.org/TR/xpath20/
  - http://www.w3.org/TR/xslt20/
  - http://www.w3.org/TR/xquery/

Here are some useful links.

References (2 of 2)

- developerWorks® XML Zone
  - http://www.ibm.com/developerworks/xml/
- Seven part article on moving to XSLT 2.0
  - http://www.ibm.com/developerworks/library/x-xslt20pt1.html
  - http://www.ibm.com/developerworks/library/x-xslt20pt2.html
  - http://www.ibm.com/developerworks/library/x-xslt20pt3.html
  - http://www.ibm.com/developerworks/library/x-xslt20pt4.html
  - http://www.ibm.com/developerworks/library/x-xslt20pt5.html
  - http://www.ibm.com/developerworks/library/x-xslt20pt6.html
  - http://www.ibm.com/developerworks/library/x-xslt20pt7.html

Here are some useful links.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_XMLFEP_Migration.ppt

This module is also available in PDF format at: ../XMLFEP_Migration.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, disclaimer, and copyright information