Package javax.xml.transform

This package defines the generic APIs for processing transformation instructions, and performing a transformation from source to result.

See:

Description

Interface Summary	
ErrorListener	To provide customized error handling, implement this interface and use the setErrorListener method to register an instance of the implmentation with the `Transformer`.
Result	An object that implements this interface contains the information needed to build a transformation result tree.
Source	An object that implements this interface contains the information needed to act as source input (XML source or transformation instructions).
SourceLocator	This interface is primarily for the purposes of reporting where an error occurred in the XML source or transformation instructions.
Templates	An object that implements this interface is the runtime representation of processed transformation instructions.
URIResolver	An object that implements this interface that can be called by the processor to turn a URI used in document(), xsl:import, or xsl:include into a Source object.

Class Summary	
OutputKeys	Provides string constants that can be used to set output properties for a Transformer, or to retrieve output properties from a Transformer or Templates object.
Transformer	An instance of this abstract class can transform a source tree into a result tree.
TransformerFactory	A TransformerFactory instance can be used to create `Transformer` and `Templates` objects.

Exception Summary	
TransformerConfigurationException	Indicates a serious configuration error.
TransformerException	This class specifies an exceptional condition that occured during the transformation process.

Error Summary	
TransformerFactoryConfigurationError	Thrown when a problem with configuration with the Transformer Factories exists.

Package javax.xml.transform Description

This package defines the generic APIs for processing transformation instructions, and performing a transformation from source to result. These interfaces have no dependencies on SAX or the DOM standard, and try to make as few assumptions as possible about the details of the source and result of a transformation. It achieves this by defining `Source` and `Result` interfaces.

To define concrete classes for the user, the API defines specializations of the interfaces found at the root level. These interfaces are found in `javax.xml.transform.sax`, `javax.xml.transform.dom`, and `javax.xml.transform.stream`.

Creating Objects

The API allows a concrete `TransformerFactory` object to be created from the static function `TransformerFactory.newInstance()`.

Specification of Inputs and Outputs

This API defines two interface objects called `Source` and `Result`. In order to pass Source and Result objects to the interfaces, concrete classes must be used. Three concrete representations are defined for each of these objects: `StreamSource` and `StreamResult`, `SAXSource` and `SAXResult`, and `DOMSource` and `DOMResult`. Each of these objects defines a FEATURE string (which is i the form of a URL), which can be passed into `TransformerFactory.getFeature(java.lang.String)` to see if the given type of Source or Result object is supported. For instance, to test if a DOMSource and a StreamResult is supported, you can apply the following test.

```
TransformerFactory tfactory = TransformerFactory.newInstance();

if (tfactory.getFeature(DOMSource.FEATURE) && tfactory.getFeature(StreamResult.
{
   ...
}
```

Qualified Name Representation

Namespaces present something of a problem area when dealing with XML objects. Qualified Names appear in XML markup as prefixed names. But the prefixes themselves do not hold identity. Rather, it is the URIs that they contextually map to that hold the identity. Therefore, when passing a Qualified Name like "xyz:foo" among Java programs, one must provide a means to map "xyz" to a namespace.

One solution has been to create a "QName" object that holds the namespace URI, as well as the prefix and local name, but this is not always an optimal solution, as when, for example, you want to use

unique strings as keys in a dictionary object. Not having a string representation also makes it difficult to specify a namespaced identity outside the context of an XML document.

In order to pass namespaced values to transformations, for instance as a set of properties to the Serializer, this specification defines that a String "qname" object parameter be passed as two-part string, the namespace URI enclosed in curly braces ({}), followed by the local name. If the qname has a null URI, then the String object only contains the local name. An application can safely check for a non-null URI by testing to see if the first character of the name is a '{' character.

For example, if a URI and local name were obtained from an element defined with <xyz:foo xmlns:xyz="http://xyz.foo.com/yada/baz.html"/>, then the Qualified Name would be "{http://xyz.foo.com/yada/baz.html}foo". Note that the prefix is lost.

Result Tree Serialization

Serialization of the result tree to a stream can be controlled with the `Transformer.setOutputProperties(java.util.Properties)` and the `Transformer.setOutputProperty(java.lang.String, java.lang.String)` methods. Strings that match the XSLT specification for xsl:output attributes can be referenced from the `OutputKeys` class. Other strings can be specified as well. If the transformer does not recognize an output key, a `IllegalArgumentException` is thrown, unless the key name is namespace qualified. Output key names that are qualified by a namespace are ignored or passed on to the serializer mechanism.

If all that is desired is the simple identity transformation of a source to a result, then `TransformerFactory` provides a `TransformerFactory.newTransformer()` method with no arguments. This method creates a Transformer that effectively copies the source to the result. This method may be used to create a DOM from SAX events or to create an XML or HTML stream from a DOM or SAX events.

Exceptions and Error Reporting

The transformation API throw three types of specialized exceptions. A `TransformerFactoryConfigurationError` is parallel to the `FactoryConfigurationError`, and is thrown when a configuration problem with the TransformerFactory exists. This error will typically be thrown when the transformation factory class specified with the "javax.xml.transform.TransformerFactory" system property cannot be found or instantiated.

A `TransformerConfigurationException` may be thrown if for any reason a Transformer can not be created. A TransformerConfigurationException may be thrown if there is a syntax error in the transformation instructions, for example when `TransformerFactory.newTransformer (javax.xml.transform.Source)` is called.

`TransformerException` is a general exception that occurs during the course of a transformation. A transformer exception may wrap another exception, and if any of the `TransformerException.printStackTrace()` methods are called on it, it will produce a list of stack dumps, starting from the most recent. The transformer exception also provides a `SourceLocator` object which indicates where in the source tree or transformation instructions the error occurred. `TransformerException.getMessageAndLocation()` may be called to get an error message with location info, and `TransformerException.getLocationAsString()` may be called to get just the location string.

Transformation warnings and errors are normally first sent to a `ErrorListener`, at which point the implementor may decide to report the error or warning, and may decide to throw an exception for a non-fatal error. The error listener may be set via `TransformerFactory.setErrorListener (javax.xml.transform.ErrorListener)` for reporting errors that have to do with syntax errors in the transformation instructions, or via `Transformer.setErrorListener (javax.xml.transform.ErrorListener)` to report errors that occur during the transformation. The error listener on both objects should always be valid and non-null, whether set by the user or a default implementation provided by the processor.

Resolution of URIs within a transformation

The API provides a way for URIs referenced from within the stylesheet instructions or within the transformation to be resolved by the calling application. This can be done by creating a class that implements the `URIResolver` interface, with its one method, `URIResolver.resolve (java.lang.String, java.lang.String)`, and use this class to set the URI resolution for the transformation instructions or transformation with `TransformerFactory.setURIResolver (javax.xml.transform.URIResolver)` or `Transformer.setURIResolver (javax.xml.transform.URIResolver)`. The `URIResolver.resolve` method takes two String arguments, the URI found in the stylesheet instructions or built as part of the transformation process, and the base URI in effect when the URI passed as the first argument was encountered. The returned `Source` object must be usable by the transformer, as specified in its implemented features.