



CHAPTER 21

javax.xml.parsers, java.xml.transform, and Subpackages

This chapter documents the `javax.xml.parsers` package and the `javax.xml.transform` package and its subpackages. These packages comprise the Java API for XML Processing, or JAXP. Before being integrated in Java 1.4, JAXP was available as a standard extension, which explains the “javax” prefix in the package names. The packages in this chapter are:

`javax.xml.parsers`

This package defines parser classes that serve as a wrapper around underlying DOM and SAX XML parsers, and also defines factory classes that are used to obtain instances of those parser classes.

`javax.xml.transform`

This package defines classes and interfaces for transforming the representation and content of an XML document with XSLT. It defines `Source` and `Result` interfaces to represent a source document and a result document. Sub-packages provide implementations of these classes that represent documents in different ways.

`javax.xml.transform.dom`

This package implements the `Source` and `Result` interfaces that represent documents as DOM document trees.

`javax.xml.transform.sax`

This package implements the `Source` and `Result` interfaces to represent documents as sequences of SAX parser events. It also defines other SAX-related transformation classes.

`javax.xml.transform.stream`

This package implements the `Source` and `Result` interfaces that represent documents as streams of text.

Package `javax.xml.parsers`

Java 1.4

This package defines classes that represent XML parsers and factory classes for obtaining instances of those parser classes. `DocumentBuilder` is a DOM-based XML parser

Package *javax.xml.parsers*

created from a `DocumentBuilderFactory`. `SAXParser` is a SAX-based XML parser created from a `SAXParserFactory`. Note that this package does not include parser implementations. Instead, it is an implementation-independent layer that supports “pluggable” XML parsers. Furthermore, this package does not define a DOM or SAX API for working with XML documents. The DOM API is defined in `org.w3c.dom`, and the SAX API is defined in `org.xml.sax` and its subpackages.

Classes:

```
public abstract class DocumentBuilder;  
public abstract class DocumentBuilderFactory;  
public abstract class SAXParser;  
public abstract class SAXParserFactory;
```

Exception:

```
public class ParserConfigurationException extends Exception;
```

Errors:

```
public class FactoryConfigurationError extends Error;
```

DocumentBuilder

Java 1.4

javax.xml.parsers

This class defines a high-level API to an underlying DOM parser implementation. Obtain a `DocumentBuilder` from a `DocumentBuilderFactory`. After obtaining a `DocumentBuilder`, you can provide `org.xml.sax.ErrorHandler` and `org.xml.sax.EntityResolver` objects, if desired. (These classes are defined by the SAX API but are useful for DOM parsers as well.) You may also want to call `isNamespaceAware()` and `isValidating()` to ensure that the parser is configured with the features your application requires. Finally, use one of the `parse()` methods to read an XML document from a stream, file, URL, or `org.xml.sax.InputSource` object, parse that document, and convert it into a `org.w3c.dom.Document` tree. Note that `DocumentBuilder` objects are not typically thread-safe.

If you want to obtain an empty `Document` object (so that you can build the document tree from scratch, for example) call `newDocument()`. Or use `getDOMImplementation()` to obtain the `org.w3c.dom.DOMImplementation` object of the underlying DOM implementation from which you can also create an empty `Document`.

See the `org.w3c.dom` package for information on what you can do with a `Document` object once you have used a `DocumentBuilder` to create it.

```
public abstract class DocumentBuilder {  
    // Protected Constructors  
    protected DocumentBuilder();  
    // Property Accessor Methods (by property name)  
    public abstract org.w3c.dom.DOMImplementation getDOMImplementation();  
    public abstract boolean isNamespaceAware();  
    public abstract boolean isValidating();  
    // Public Instance Methods  
    public abstract org.w3c.dom.Document newDocument();  
    public abstract org.w3c.dom.Document parse(org.xml.sax.InputSource is) throws org.xml.sax.SAXException,  
        java.io.IOException;  
    public org.w3c.dom.Document parse(java.io.InputStream is) throws org.xml.sax.SAXException, java.io.IOException;  
    public org.w3c.dom.Document parse(String uri) throws org.xml.sax.SAXException, java.io.IOException;  
    public org.w3c.dom.Document parse(java.io.File f) throws org.xml.sax.SAXException, java.io.IOException;  
    public org.w3c.dom.Document parse(java.io.InputStream is, String systemId) throws org.xml.sax.SAXException,  
        java.io.IOException;
```

```
public abstract void setEntityResolver(org.xml.sax.EntityResolver er);
public abstract void setErrorHandler(org.xml.sax.ErrorHandler eh);
}
```

Returned By: DocumentBuilderFactory.newDocumentBuilder()

DocumentBuilderFactory

Java 1.4

javax.xml.parsers

A DocumentBuilderFactory is a factory class for creating DocumentBuilder objects. You can obtain a DocumentBuilderFactory by instantiating an implementation-specific subclass provided by a parser vendor, but it is much more common to simply call newInstance() to obtain an instance of the factory that has been configured as the default for the system. Once you have obtained a factory object, you can use the various set methods to configure the properties of the DocumentBuilder objects it will create. These methods allow you to specify whether the parsers created by the factory will:

- Coalesce CDATA sections with adjacent text nodes
- Expand entity references or leave them unexpanded in the document tree
- Omit XML comments from the document tree
- Omit ignorable whitespace from the document tree
- Handle XML namespaces correctly
- Validate XML documents against a DTD or other schema

In addition to the various implementation-independent set methods, you can also use setAttribute() pass an implementation-dependent named attribute to the underlying parser implementation. Once you have configured the factory object as desired, simply call newDocumentBuilder() to create a DocumentBuilder object with the all of the attributes you have specified. Note that DocumentBuilderFactory objects are not typically thread-safe.

The javax.xml.parsers package allows parser implementations to be “plugged in.” This pluggability is provided by the newInstance() method, which follows the following steps to determine which DocumentBuilderFactory implementation to use:

- If the javax.xml.parsers.DocumentBuilderFactory system property is defined, then the class specified by that property is used.
- Otherwise, if the jre/lib/jaxp.properties file exists in the Java distribution and contains a definition for the javax.xml.parsers.DocumentBuilderFactory property, then the class specified by that property is used.
- Otherwise, if any of the JAR files on the classpath includes a file named META-INF/services/javax.xml.parsers.DocumentBuilderFactory, then the class named in that file is used.
- Otherwise, a default implementation provided by the Java implementation is used.

```
public abstract class DocumentBuilderFactory {
    // Protected Constructors
    protected DocumentBuilderFactory();
    // Public Class Methods
    public static DocumentBuilderFactory newInstance() throws FactoryConfigurationError;
    // Property Accessor Methods (by property name)
    public boolean isCoalescing();
}
```

DocumentBuilderFactory

```
public void setCoalescing(boolean coalescing);
public boolean isExpandEntityReferences();
public void setExpandEntityReferences(boolean expandEntityRef);
public boolean isIgnoringComments();
public void setIgnoringComments(boolean ignoreComments);
public boolean isIgnoringElementContentWhitespace();
public void setIgnoringElementContentWhitespace(boolean whitespace);
public boolean isNamespaceAware();
public void setNamespaceAware(boolean awareness);
public boolean isValidating();
public void setValidating(boolean validating);
// Public Instance Methods
public abstract Object getAttribute(String name) throws IllegalArgumentException;
public abstract DocumentBuilder newDocumentBuilder() throws ParserConfigurationException;
public abstract void setAttribute(String name, Object value) throws IllegalArgumentException;
}
```

Returned By: DocumentBuilderFactory.newInstance()

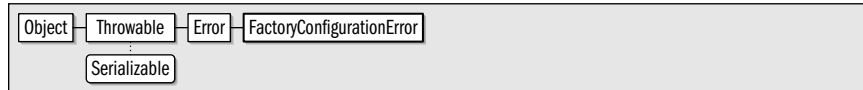
FactoryConfigurationError

Java 1.4

javax.xml.parsers

serializable error

This signals a nonrecoverable problem instantiating a parser factory. This usually means that a pluggable parser implementation has been incorrectly plugged in and the newInstance() method cannot locate the specified factory implementation class.



```
public class FactoryConfigurationError extends Error {
// Public Constructors
public FactoryConfigurationError();
public FactoryConfigurationError(Exception e);
public FactoryConfigurationError(String msg);
public FactoryConfigurationError(Exception e, String msg);
// Public Instance Methods
public Exception getException(); default:null
// Public Methods Overriding Throwable
public String getMessage(); default:null
}
```

Thrown By: DocumentBuilderFactory.newInstance(), SAXParserFactory.newInstance()

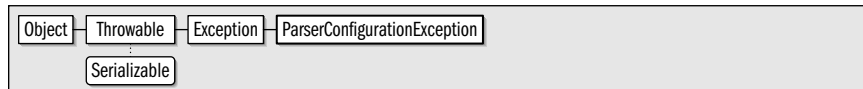
ParserConfigurationException

Java 1.4

javax.xml.parsers

serializable checked

This exception signals a parser configuration problem that prevents a parser factory object from creating a parser object.



```
public class ParserConfigurationException extends Exception {
// Public Constructors
public ParserConfigurationException();
}
```

```
public ParserConfigurationException(String msg);
}
```

Thrown By: DocumentBuilderFactory.newDocumentBuilder(), SAXParserFactory.{getFeature(), newSAXParser(), setFeature()}

SAXParser

Java 1.4

javax.xml.parsers

The `SAXParser` class is a wrapper around an `org.xml.sax.XMLReader` class and is used to parse XML documents using the SAX version 2 API. Obtain a `SAXParser` from a `SAXParserFactory`. Call `setProperty()` if desired to set a property on the underlying parser. (See <http://www.saxproject.org> for a description of standard SAX properties and their values.) Finally, call one of the `parse()` methods to parse an XML document from a stream, file, URL, or `org.xml.sax.InputSource`. The SAX API is an event-driven one. A SAX parser does not build a document tree to describe an XML document like a DOM parser does. Instead, it describes the XML document to your application by invoking methods on an object the application provides. This is the purpose of the `org.xml.sax.helpers.DefaultHandler` object that is passed to the `parse()` method: you subclass this class to implement the methods you care about, and the parser will invoke those methods at appropriate times. For example, when the parser encounters an XML tag in a document, it parses the tag, and calls the `startElement()` method to tell you about it. And when it finds a run of plain text, it passes that text to the `characters()` method.

Instead of using one of the `parse()` methods of this class, you can also call `getXMLReader()` to obtain the underlying `XMLReader` object and work with it directly to parse the desired document. `SAXParser` objects are not typically thread-safe.

Note that the `getParser()` method as well as the `parse()` methods that take an `org.xml.sax.HandlerBase` object are based on the SAX version 1 API and should be avoided.

```
public abstract class SAXParser {
    // Protected Constructors
    protected SAXParser();

    // Property Accessor Methods (by property name)
    public abstract boolean isNamespaceAware();
    public abstract org.xml.sax.Parser getParser() throws org.xml.sax.SAXException;
    public abstract boolean isValidating();
    public abstract org.xml.sax.XMLReader getXMLReader() throws org.xml.sax.SAXException;

    // Public Instance Methods
    public abstract Object getProperty(String name) throws org.xml.sax.SAXNotRecognizedException,
        org.xml.sax.SAXNotSupportedException;
    public void parse(java.io.File f, org.xml.sax.helpers.DefaultHandler dh) throws org.xml.sax.SAXException,
        java.io.IOException;
    public void parse(org.xml.sax.InputSource is, org.xml.sax.HandlerBase hb) throws org.xml.sax.SAXException,
        java.io.IOException;
    public void parse(org.xml.sax.InputSource is, org.xml.sax.helpers.DefaultHandler dh)
        throws org.xml.sax.SAXException, java.io.IOException;
    public void parse(java.io.InputStream is, org.xml.sax.helpers.DefaultHandler dh) throws org.xml.sax.SAXException,
        java.io.IOException;
    public void parse(java.io.InputStream is, org.xml.sax.HandlerBase hb) throws org.xml.sax.SAXException,
        java.io.IOException;
    public void parse(String uri, org.xml.sax.HandlerBase hb) throws org.xml.sax.SAXException, java.io.IOException;
    public void parse(String uri, org.xml.sax.helpers.DefaultHandler dh) throws org.xml.sax.SAXException,
        java.io.IOException;
}
```

SAXParser

```
public void parse(java.io.File f, org.xml.sax.HandlerBase hb) throws org.xml.sax.SAXException, java.io.IOException;
public void parse(java.io.InputStream is, org.xml.sax.HandlerBase hb, String systemId)
    throws org.xml.sax.SAXException, java.io.IOException;
public void parse(java.io.InputStream is, org.xml.sax.helpers.DefaultHandler dh, String systemId)
    throws org.xml.sax.SAXException, java.io.IOException;
public abstract void setProperty(String name, Object value) throws org.xml.sax.SAXNotRecognizedException,
    org.xml.sax.SAXNotSupportedException;
}
```

Returned By: SAXParserFactory.newSAXParser()

SAXParserFactory

Java 1.4

javax.xml.parsers

This class is a factory for SAXParser objects. Obtain a SAXParserFactory by calling the `newInstance()` method which instantiates the default SAXParserFactory subclass provided with your Java implementation, or instantiates some other SAXParserFactory that has been “plugged in.”

Once you have a SAXParserFactory object, you can use `setValidating()` and `setNamespaceAware()` to specify whether the parsers it creates will be validating parsers or not and whether they will know how to handle XML namespaces. You may also call `setFeature()` to set a feature of the underlying parser implementation. See <http://www.saxproject.org> for the names of standard parser features that can be enabled and disabled with this method.

Once you have created and configured your factory object, simply call `newSAXParser()` to create a SAXParser object. Note that SAXParserFactory implementations are not typically thread-safe.

The `javax.xml.parsers` package allows parser implementations to be “plugged in.” This pluggability is provided by the `newInstance()` method, which follows the following steps to determine which SAXParserFactory subclass to use:

- If the `javax.xml.parsers.SAXParserFactory` system property is defined, then the class specified by that property is used.
- Otherwise, if the `jre/lib/jaxp.properties` file exists in the Java distribution and contains a definition for the `javax.xml.parsers.SAXParserFactory` property, then the class specified by that property is used.
- Otherwise, if any of the JAR files on the classpath includes a file named `META-INF/services/javax.xml.parsers.SAXParserFactory`, then the class named in that file is used.
- Otherwise, a default implementation provided by the Java platform is used.

```
public abstract class SAXParserFactory {
    // Protected Constructors
    protected SAXParserFactory();
    // Public Class Methods
    public static SAXParserFactory newInstance() throws FactoryConfigurationError;
    // Public Instance Methods
    public abstract boolean getFeature(String name) throws ParserConfigurationException,
        org.xml.sax.SAXNotRecognizedException, org.xml.sax.SAXNotSupportedException;
    public boolean isNamespaceAware();
    public boolean isValidating();
    public abstract SAXParser newSAXParser() throws ParserConfigurationException, org.xml.sax.SAXException;
}
```

```
public abstract void setFeature(String name, boolean value) throws ParserConfigurationException,  
    org.xml.sax.SAXNotRecognizedException, org.xml.sax.SAXNotSupportedException;  
public void setNamespaceAware(boolean awareness);  
public void setValidating(boolean validating);  
}
```

Returned By: SAXParserFactory.newInstance()

Package javax.xml.transform

Java 1.4

This package defines a high-level, implementation-independent API for using an XSLT engine or other document transformation system for transforming XML document content, and also for transforming XML documents from one form (such as a stream of text in a file) to another form (such as a tree of DOM nodes). The **Source** interface is a very generic description of a document source. Three concrete implementations that represent documents in text form, as DOM trees, and as sequences of SAX parser events are defined in the three sub-packages of this package. The **Result** interface is a similarly high-level description of what form the source document should be transformed into. The three sub-packages define three **Result** implementations that represent XML documents as streams or files, as DOM trees, and as sequences of SAX parser events.

The **TransformerFactory** class represents the document transformation engine. The implementation provides a default factory that represents an XSLT engine. A **TransformerFactory** can be used to produce **Templates** objects that represent compiled XSL stylesheets (or other implementation-dependent forms of transformation instructions). Documents are actually transformed from **Source** to **Result** with a **Transformer** object, which is obtained from a **Templates** object, or directly from a **TransformerFactory**.

Interfaces:

```
public interface ErrorListener;  
public interface Result;  
public interface Source;  
public interface SourceLocator;  
public interface Templates;  
public interface URIResolver;
```

Classes:

```
public class OutputKeys;  
public abstract class Transformer;  
public abstract class TransformerFactory;
```

Exceptions:

```
public class TransformerException extends Exception;  
    public class TransformerConfigurationException extends TransformerException;
```

Errors:

```
public class TransformerFactoryConfigurationError extends Error;
```

ErrorListener

Java 1.4

javax.xml.transform

This interface defines methods that `Transformer` and `TransformerFactory` use for reporting warnings, errors, and fatal errors to an application. To use an `ErrorListener`, an application must implement this interface and pass an implementing object to the `setErrorListener()` method of `Transformer` or `TransformerFactory`. The argument to each method of this interface is a `TransformerException` object, and the implementation of these methods can throw that exception if it chooses, or it can simply log the warning or error in some way and return. A `Transformer` or `TransformerFactory` is not required to continue processing after reporting a non-recoverable error with an invocation of the `fatalError()` method.

If you are familiar with the SAX API for parsing XML documents, you'll recognize that this interface is very similar to `org.xml.sax.ErrorHandler`.

```
public interface ErrorListener {
    // Public Instance Methods
    public abstract void error(TransformerException exception) throws TransformerException;
    public abstract void fatalError(TransformerException exception) throws TransformerException;
    public abstract void warning(TransformerException exception) throws TransformerException;
}
```

Passed To: `Transformer.setErrorListener()`, `TransformerFactory.setErrorListener()`

Returned By: `Transformer.getErrorListener()`, `TransformerFactory.getErrorListener()`

OutputKeys

Java 1.4

javax.xml.transform

This class defines string constants that hold the names of the attributes of an `<xsl:output>` tag in an XSLT stylesheet. These are also legal key values for the `Properties` object returned by `Templates.getOutputProperties()` and passed to `Transformer.setOutputProperties()`.

```
public class OutputKeys {
    // No Constructor
    // Public Constants
    public static final String CDATA_SECTION_ELEMENTS;           = [quot ] cdata-section-elements [quot ]
    public static final String DOCTYPE_PUBLIC;                  = [quot ] doctype-public [quot ]
    public static final String DOCTYPE_SYSTEM;                  = [quot ] doctype-system [quot ]
    public static final String ENCODING;                         = [quot ] encoding [quot ]
    public static final String INDENT;                           = [quot ] indent [quot ]
    public static final String MEDIA_TYPE;                       = [quot ] media-type [quot ]
    public static final String METHOD;                         = [quot ] method [quot ]
    public static final String OMIT_XML_DECLARATION;             = [quot ] omit-xml-declaration [quot ]
    public static final String STANDALONE;                       = [quot ] standalone [quot ]
    public static final String VERSION;                         = [quot ] version [quot ]
}
```

Result

Java 1.4

javax.xml.transform

This interface represents, in a very general way, the result of an XML transformation. `setSystemId()` specifies the system identifier of the result as a URL. This is useful when the result is to be written as a file, but it can also be useful for error reporting and for resolution of relative URLs even when the `Result` object does not represent a file. All other methods related to the result are the responsibility of the concrete implementation of this interface. See the `DOMResult`, `SAXResult` and `StreamResult` implementations in the three subpackages of this package.


```

public interface Result {
// Public Constants
    public static final String          = [quot ] javax.xml.transform.disable-output-escaping [quot ]
    PI_DISABLE_OUTPUT_ESCAPING;
    public static final String          = [quot ] javax.xml.transform.enable-output-escaping [quot ]
    PI_ENABLE_OUTPUT_ESCAPING;
// Public Instance Methods
    public abstract String getSystemId();
    public abstract void setSystemId(String systemId);
}

```

Implementations: javax.xml.transform.dom.DOMResult, javax.xml.transform.sax.SAXResult, javax.xml.transform.stream.StreamResult

Passed To: Transformer.transform(), javax.xml.transform.sax.TransformerHandler.setResult()

Source

Java 1.4

javax.xml.transform

This interface represents, in a very general way, the source of an XML document. `setSystemId()` specifies the system identifier of the document in the form of a URL. This is useful for resolving relative URLs and for error reporting even when the document is not read directly from a URL. All other methods related to the document source are the responsibility of the concrete implementation of this interface. See the `DOMSource`, `SAXSource` and `StreamSource` implementations in the three sub-packages of this package.

```

public interface Source {
// Public Instance Methods
    public abstract String getSystemId();
    public abstract void setSystemId(String systemId);
}

```

Implementations: javax.xml.transform.dom.DOMSource, javax.xml.transform.sax.SAXSource, javax.xml.transform.stream.StreamSource

Passed To: Transformer.transform(), TransformerFactory.{getAssociatedStylesheet(), newTemplates(), newTransformer()}, javax.xml.transform.sax.SAXSource.sourceToInputSource(), javax.xml.transform.sax.SAXTransformerFactory.{newTransformerHandler(), newXMLFilter()}

Returned By: TransformerFactory.getAssociatedStylesheet(), URIResolver.resolve()

SourceLocator

Java 1.4

javax.xml.transform

This interface defines methods that return the system and public identifiers of an XML document, and return a line number and column number within that document. `SourceLocator` objects are used with `TransformerException` and `TransformerConfigurationException` objects to specify the location in an XML file at which the exception occurred. Note, however that system and public identifiers are not always available for a document, and so `getSystemId()` and `getPublicId()` may return null. Also, a `Transformer` is not required to track line and column numbers precisely, or at all, so `getLineNumber()` and `getColumnNumber()` may return -1 to indicate that line and column number information is not available. If they return a value other than -1, it should be considered an approximation to the actual value. Note that lines and columns within a document are numbered starting with 1, not with 0.

If you are familiar with the SAX API for parsing XML, you'll recognize this interface as a renamed version of `org.xml.sax.Locator`.

SourceLocator

```
public interface SourceLocator {  
    // Public Instance Methods  
    public abstract int getColumnNumber();  
    public abstract int getLineNumber();  
    public abstract String getPublicId();  
    public abstract String getSystemId();  
}
```

Implementations: javax.xml.transform.dom.DOMLocator

Passed To: TransformerConfigurationException.TransformerConfigurationException(),
TransformerException.{setLocator(), TransformerException()}

Returned By: TransformerException.getLocator()

Templates

Java 1.4

javax.xml.transform

This interface represents a set of transformation instructions for transforming a **Source** document into a **Result** document. The javax.xml.transform package is theoretically independent of type of transformation, but in practice, an object of this type always represents the compiled form of an XSLT stylesheet. Obtain a **Templates** object from a **TransformerFactory** object, or with a javax.xml.transform.sax.TemplatesHandler. Once you have a **Templates** object, you can use the **newTransformer()** method to create a **Transformer** object for applying the templates to a **Source** to produce a **Result** document.

getOutputProperties() returns a java.util.Properties object that defines name/value pairs specifying details about how a textual version of the **Result** document should be produced. These properties are specified in an XSLT stylesheet with the <xsl:output> element. The constants defined by the **OutputKeys** are legal output property names. The returned **Properties** object contains explicitly properties directly, and contains default values in a parent **Properties** object. This means that if you query a property value with **getProperty()**, you'll get an explicitly specified value or a default value. On the other hand, if you query a property with the **get()** method (inherited by **Properties** from its superclass) you'll get a property value if it was explicitly specified in the stylesheet, or null if it was not specified. The returned **Properties** object is a clone of the internal value, so you can modify it (before passing it to the **setOutputProperties()** method of a **Transformer** object, for example) without affecting the **Templates** object.

Templates implementations are required to be thread-safe. A **Templates** object can be used to create any number of **Transformer** objects.

```
public interface Templates {  
    // Public Instance Methods  
    public abstract java.util.Properties getOutputProperties();  
    public abstract Transformer newTransformer() throws TransformerConfigurationException;  
}
```

Passed To: javax.xml.transform.sax.SAXTransformerFactory.{newTransformerHandler(), newXMLFilter()}

Returned By: TransformerFactory.newTemplates(),
javax.xml.transform.sax.TemplatesHandler.getTemplates()

Transformer

Java 1.4

javax.xml.transform

Objects of this type are used to transform a **Source** document into a **Result** document. Obtain a **Transformer** object from a **TransformerFactory** object, from a **Templates** object cre-

ated by a `TransformerFactory`, or from a `TransformerHandler` object created by a `SAXTransformerFactory` (these last two types are from the `javax.xml.transform.sax` package).

Once you have a `Transformer` object, you may need to configure it before using it to transform documents. `setErrorListener()` and `setURIResolver()` allow you to specify `ErrorListener` and `URIResolver` object that the `Transformer` can use. `setOutputProperty()` and `setOutputProperties()` allow you to specify name/value pairs that affect the text formatting of the `Result` document (if that document is written out in text format). `OutputKeys` defines constants that represent the set of standard output property names. The output properties you specify with these methods override any output properties specified (with an `<xsl:output>` tag) in the `Templates` object. Use `setParameter()` to supply values for any top-level parameters defined (with `<xsl:param>` tags) in the stylesheet. Note that if the name of any such parameter is a qualified name, then it appears in the stylesheet with a namespace prefix. You can't use the prefix with the `setParameter()` method, however, and you must instead specify the parameter name using the URI of the namespace within curly braces followed by the local name. If no namespace is involved, then you can just use the simple name of the parameter with no curly braces or URIs.

Once you have created and configured a `Transformer` object, use the `transform()` method to perform a document transformation. This method transforms the specified `Source` document and creates the transformed document specified by the `Result` object.

`Transformer` implementations are not typically thread-safe. You can reuse a `Transformer` object and call `transform()` any number of times (just not concurrently). The output properties and parameters you specify are not changed by calling the `transform()` method and can be reused.

```
public abstract class Transformer {
    // Protected Constructors
    protected Transformer();
    // Property Accessor Methods (by property name)
    public abstract ErrorListener getErrorListener();
    public abstract void setErrorListener(ErrorListener listener) throws IllegalArgumentException;
    public abstract java.util.Properties getOutputProperties();
    public abstract void setOutputProperties(java.util.Properties oformat) throws IllegalArgumentException;
    public abstract URIResolver getURIResolver();
    public abstract void setURIResolver(URIResolver resolver);
    // Public Instance Methods
    public abstract void clearParameters();
    public abstract String getOutputProperty(String name) throws IllegalArgumentException;
    public abstract Object getParameter(String name);
    public abstract void setOutputProperty(String name, String value) throws IllegalArgumentException;
    public abstract void setParameter(String name, Object value);
    public abstract void transform(Source xmlSource, Result outputTarget) throws TransformerException;
}
```

Returned By: `Templates.newTransformer()`, `TransformerFactory.newTransformer()`,
`javax.xml.transform.sax.TransformerHandler.getTransformer()`

TransformerConfigurationException

Java 1.4

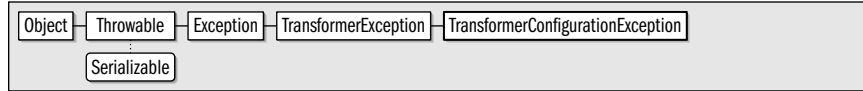
`javax.xml.transform`

serializable checked

This exception signals a problem creating a `Transformer` object. This may occur, for example, if there is a syntax error in the XSL stylesheet that contains the transformation

TransformerConfigurationException

instructions. Use the inherited `getLocator()` method to obtain a `SourceLocator` that describes the document location at which the exception occurred.



```
public class TransformerConfigurationException extends TransformerException {  
    // Public Constructors  
    public TransformerConfigurationException();  
    public TransformerConfigurationException(Throwable e);  
    public TransformerConfigurationException(String msg);  
    public TransformerConfigurationException(String message, SourceLocator locator);  
    public TransformerConfigurationException(String msg, Throwable e);  
    public TransformerConfigurationException(String message, SourceLocator locator, Throwable e);  
}
```

Thrown By: `Templates.newTransformer()`, `TransformerFactory.getAssociatedStylesheet()`, `newTemplates()`, `newTransformer()`, `javax.xml.transform.sax.SAXTransformerFactory.newTemplatesHandler()`, `newTransformerHandler()`, `newXMLFilter()`

TransformerException

Java 1.4

`javax.xml.transform`

serializable checked

This exception signals a problem while reading or transforming a document. Call `getLocator()` to obtain a `SourceLocator` object that describes the document location where the exception occurred.



```
public class TransformerException extends Exception {  
    // Public Constructors  
    public TransformerException(String message);  
    public TransformerException(Throwable e);  
    public TransformerException(String message, Throwable e);  
    public TransformerException(String message, SourceLocator locator);  
    public TransformerException(String message, SourceLocator locator, Throwable e);  
    // Public Instance Methods  
    public Throwable getException();  
    public String getLocationAsString();  
    public SourceLocator getLocator();  
    public String getMessageAndLocation();  
    public void setLocator(SourceLocator location);  
    // Public Methods Overriding Throwable  
    public Throwable getCause();  
    public Throwable initCause(Throwable cause);  
    public void printStackTrace();  
    public void printStackTrace(java.io.PrintStream s);  
    public void printStackTrace(java.io.PrintWriter s);  
}
```

Subclasses: `TransformerConfigurationException`

Passed To: `ErrorListener.error()`, `fatalError()`, `warning()`

Thrown By: `ErrorListener.{error(), fatalError(), warning()}`, `Transformer.transform()`, `URIResolver.resolve()`

TransformerFactory

Java 1.4

`javax.xml.transform`

An instance of this abstract class represents a document “transformation engine” such as an XSLT processor. A `TransformerFactory` is used to create `Transformer` objects that perform document transformations, and can also be used to process transformation instructions (such as XSLT stylesheets) into compiled `Templates` objects.

Obtain a `TransformerFactory` instance by calling the static `newInstance()` method. `newInstance()` returns an instance of the default implementation for your Java installation, or, if the system property `javax.xml.transform.TransformerFactory` is set, then it returns an instance of the implementation class named by that property. The default `TransformerFactory` implementation provided with the Java distribution transforms XML documents using XSL stylesheets.

You can configure a `TransformerFactory` instance by calling `setErrorListener()` and `setURIResolver()` to specify an `ErrorListener` object and a `URIResolver` object to be used by the factory when reading and parsing XSL stylesheets. The `setAttribute()` and `getAttribute()` methods can be used to set and query implementation-dependent attributes of the transformation engine. The default engine supplied by Sun does not define any attributes. The `getFeature()` method is used to test whether the factory supports a given feature. For uniqueness, feature names are expressed as URIs, and each of the `Source` and `Result` implementations defined in the three subpackages of this package define a `FEATURE` constant that specifies a URL that you can use to test whether a `TransformerFactory` supports that particular `Source` or `Result` type.

Once you have obtained and configured your `TransformerFactory` object, you can use it in several ways. If you call the `newTransformer()` method that takes no arguments, you’ll obtain a `Transformer` object that transforms the format or representation of an XML document without transforming its content. For example, you could use a `Transformer` created in this way to transform a DOM tree (represented by a `javax.xml.transform.dom.DOMSource` object) to a stream of XML text stored in a file named by a `javax.xml.transform.stream.StreamResult`.

Another way to use a `TransformerFactory` is to call the `newTemplates()` method, passing in a `Source` object that represents an XSL stylesheet. This produces a `Templates` object, which you can use to obtain a `Transformer` object that applies the stylesheet to transform document content. Alternatively, if you do not plan to create more than one `Transformer` object from the `Templates` object, you can combine the two steps and simply pass the `Source` object representing the stylesheet to the one-argument version of `newTransformer()`.

XML documents may include references to XSL stylesheets in the form of an `xml-stylesheet` processing instruction. The `getAssociatedStylesheet()` method reads the XML document represented by a `Source` object and returns a new `Source` object that represents the stylesheet (or the concatenation of all the stylesheets) contained in that document that match the media, title, and charset constraints defined by the other three parameters (which may be null). If you want to process an XML document using the stylesheet that it defines itself, use this method to obtain a `Source` object that you can pass to `newTransformer()` to create the `Transformer` object that you can use to transform the document.

Typically, `TransformerFactory` implementations are not thread-safe.

```
public abstract class TransformerFactory {
    // Protected Constructors
```

TransformerFactory

```
protected TransformerFactory();  
// Public Class Methods  
public static TransformerFactory newInstance() throws TransformerFactoryConfigurationError;  
// Public Instance Methods  
public abstract Source getAssociatedStylesheet(Source source, String media, String title, String charset)  
    throws TransformerConfigurationException;  
public abstract Object getAttribute(String name) throws IllegalArgumentException;  
public abstract ErrorListener getErrorListener();  
public abstract boolean getFeature(String name);  
public abstract URIResolver getURIResolver();  
public abstract Templates newTemplates(Source source) throws TransformerConfigurationException;  
public abstract Transformer newTransformer() throws TransformerConfigurationException;  
public abstract Transformer newTransformer(Source source) throws TransformerConfigurationException;  
public abstract void setAttribute(String name, Object value) throws IllegalArgumentException;  
public abstract void setErrorListener(ErrorListener listener) throws IllegalArgumentException;  
public abstract void setURIResolver(URIResolver resolver);  
}
```

Subclasses: javax.xml.transform.sax.SAXTransformerFactory

Returned By: TransformerFactory.newInstance()

TransformerFactoryConfigurationError

Java 1.4

javax.xml.transform

serializable error

This error class signals a fatal problem while creating a **TransformerFactory**. It usually signals a configuration problem, such as the system property `javax.xml.transform.TransformerFactory` has a value that is not a valid classname, or that the class path does not contain the specified factory implementation class.



```
public class TransformerFactoryConfigurationError extends Error {  
// Public Constructors  
public TransformerFactoryConfigurationError();  
public TransformerFactoryConfigurationError(String msg);  
public TransformerFactoryConfigurationError(Exception e);  
public TransformerFactoryConfigurationError(Exception e, String msg);  
// Public Instance Methods  
public Exception getException(); default:null  
// Public Methods Overriding Throwable  
public String getMessage(); default:null  
}
```

Thrown By: TransformerFactory.newInstance()

URIResolver

Java 1.4

javax.xml.transform

This interface allows an application to tell a **Transformer** how to resolve the URIs that appear in an XSLT stylesheet. If you pass a **URIResolver** to the `setURIResolver()` method of a **Transformer** or **TransformerFactory**, and then the **Transformer** or **TransformerFactory** encounters a URI, it first passes that URI, along with the base URI to the `resolve()` method of the **URIResolver**. If `resolve()` returns a **Source** object, then the **Transformer** will use that **Source**. If a **Transformer** or **TransformerFactory** has no **URIResolver** registered, or if the `resolve()` method returns null, then the transformer or factory will attempt to resolve the URI itself.

```
public interface URIResolver {
// Public Instance Methods
    public abstract Source resolve(String href, String base) throws TransformerException;
}
```

Passed To: Transformer.setURIResolver(), TransformerFactory.setURIResolver()

Returned By: Transformer.getURIResolver(), TransformerFactory.getURIResolver()

Package javax.xml.transform.dom

Java 1.4

This package contains Source and Result implementations that work with DOM document trees and subtrees.

Interfaces:

public interface **DOMLocator** extends javax.xml.transform.SourceLocator;

Classes:

public class **DOMResult** implements javax.xml.transform.Result;

public class **DOMSource** implements javax.xml.transform.Source;

DOMLocator

Java 1.4

javax.xml.transform.dom

This class extends SourceLocator to define a method for retrieving a DOM Node object, which is typically used to indicate the source of an error in the transformation process. See SourceLocator and TransformerException.

SourceLocator → DOMLocator

```
public interface DOMLocator extends javax.xml.transform.SourceLocator {
// Public Instance Methods
    public abstract org.w3c.dom.Node getOriginatingNode();
}
```

DOMResult

Java 1.4

javax.xml.transform.dom

This class is a Result implementation that writes XML content by generating a DOM tree to represent that content. If you pass an org.w3c.dom.Node to the constructor or to setNode(), the DOMResult will create the result tree as a child of the specified node (which should typically be a Document or Element node). If you do not specify a node, the DOMResult will create a new Document node when it creates the result tree. You can retrieve this Document with getNode().

Object → DOMResult → Result

```
public class DOMResult implements javax.xml.transform.Result {
// Public Constructors
    public DOMResult();
    public DOMResult(org.w3c.dom.Node node);
    public DOMResult(org.w3c.dom.Node node, String systemID);
}
```

DOMResult

```
// Public Constants
public static final String FEATURE;          = [quot ] http://javax.xml.transform.dom.DOMResult/feature [quot ]
// Public Instance Methods
public org.w3c.dom.Node getNode();                                default:null
public void setNode(org.w3c.dom.Node node);
// Methods Implementing Result
public String getSystemId();                                      default:null
public void setSystemId(String systemId);
}
```

DOMSource

Java 1.4

javax.xml.transform.dom

This class is a *Source* implementation that reads an XML document from a DOM document tree or subtree. Pass the *org.w3c.dom.Node* object that represents the root of the tree or subtree to the constructor or to *setNode()*. When possible, it is also useful to provide a system id (a filename or URL) for use in error messages and for resolving relative URLs contained in the document.

Object	DOMSource	Source
--------	-----------	--------

```
public class DOMSource implements javax.xml.transform.Source {
// Public Constructors
public DOMSource();
public DOMSource(org.w3c.dom.Node n);
public DOMSource(org.w3c.dom.Node node, String systemId);
// Public Constants
public static final String FEATURE;          = [quot ] http://javax.xml.transform.dom.DOMSource/feature [quot ]
// Public Instance Methods
public org.w3c.dom.Node getNode();                                default:null
public void setNode(org.w3c.dom.Node node);
// Methods Implementing Source
public String getSystemId();                                      default:null
public void setSystemId(String baseId);
}
```

Package javax.xml.transform.sax

Java 1.4

This package defines *Source* and *Result* implementations that work with SAX events. In addition, it includes an extension to the *TransformerFactory* class that has additional methods for returning *TemplatesHandler* and *TransformerHandler* objects. These objects implement SAX handler interfaces and are able to work with a SAX parser object to turn a series of SAX parse events into a *Templates* object or into a *Result* document. *SAXSource* and *SAXResult* adapt the *org.xml.sax* framework for use in the *javax.xml.transform* framework. By contrast, *SAXTransformerFactory*, *TemplatesHandler*, and *TransformerHandler* adapt the *javax.xml.transform* framework for use within the *org.xml.sax* parsing framework.

Interfaces:

```
public interface TemplatesHandler extends org.xml.sax.ContentHandler;
public interface TransformerHandler
    extends org.xml.sax.ContentHandler, org.xml.sax.DTDHandler, org.xml.sax.ext.LexicalHandler;
```


Classes:

public class **SAXResult** implements `javax.xml.transform.Result`;
 public class **SAXSource** implements `javax.xml.transform.Source`;
 public abstract class **SAXTransformerFactory** extends `javax.xml.transform.TransformerFactory`;

SAXResult

Java 1.4

`javax.xml.transform.sax`

This class is a `Result` implementation that describes the content of a transformed document by triggering the methods of the specified `ContentHandler`. That is, a **SAXResult** acts like a `org.xml.sax.XMLReader` object, invoking the methods of the specified `org.xml.sax.ContentHandler` object as it parses the transformed document. You may also provide a `org.xml.sax.ext.LexicalHandler` object whose methods will be invoked by the **SAXResult** by calling `setLexicalHandler()`, or by supplying a `ContentHandler` object that also implements the `LexicalHandler` interface.

Object	SAXResult	Result
--------	-----------	--------

```

public class SAXResult implements javax.xml.transform.Result {
// Public Constructors
    public SAXResult();
    public SAXResult(org.xml.sax.ContentHandler handler);
// Public Constants
    public static final String FEATURE;          = [quot ] http://javax.xml.transform.sax.SAXResult/feature [quot ]
// Public Instance Methods
    public org.xml.sax.ContentHandler getHandler();                                default:null
    public org.xml.sax.ext.LexicalHandler getLexicalHandler();                    default:null
    public void setHandler(org.xml.sax.ContentHandler handler);
    public void setLexicalHandler(org.xml.sax.ext.LexicalHandler handler);
// Methods Implementing Result
    public String getSystemId();                                                  default:null
    public void setSystemId(String systemId);
}
  
```

SAXSource

Java 1.4

`javax.xml.transform.sax`

This class is a `Source` implementation that describes a document represented as a series of SAX event method calls. A **SAXSource** requires an `org.xml.sax.InputSource` object that describes the stream to parse, and may optionally specify the `org.xml.sax.XMLReader` or `org.xml.sax.XMLFilter` that generates the SAX events. (If no `XMLReader` or `XMLFilter` is specified, then the `Transformer` object will create a default `XMLReader`.) Note that since an `InputSource` is required, a **SAXSource** does not behave significantly differently than a `StreamSource` unless an `XMLFilter` is used.

SAXSource also has one static method, `sourceToInputSource()`, which returns a `SAX InputSource` method derived from the specified `Source` object, or null if the specified `Source` cannot be converted to an `InputSource`.

Object	SAXSource	Source
--------	-----------	--------

```

public class SAXSource implements javax.xml.transform.Source {
// Public Constructors
    public SAXSource();
    public SAXSource(org.xml.sax.InputSource inputSource);
}
  
```

SAXSource

```
public SAXSource(org.xml.sax.XMLReader reader, org.xml.sax.InputSource inputSource);
// Public Constants
public static final String FEATURE;           = [quot ] http://javax.xml.transform.sax.SAXSource/feature [quot ]
// Public Class Methods
public static org.xml.sax.InputSource sourceToInputSource(javax.xml.transform.Source source);
// Public Instance Methods
public org.xml.sax.InputSource getInputSource();                                default:null
public org.xml.sax.XMLReader getXMLReader();                                default:null
public void setInputSource(org.xml.sax.InputSource inputSource);
public void setXMLReader(org.xml.sax.XMLReader reader);
// Methods Implementing Source
public String getSystemId();                                default:null
public void setSystemId(String systemId);
}
```

SAXTransformerFactory

Java 1.4

javax.xml.transform.sax

This class extends `TransformerFactory` to define additional factory methods that are useful when working with documents that are represented as sequences of SAX events. Pass the `FEATURE` constant to the `getFeature()` method of your `TransformerFactory` object to determine whether the `newTemplatesHandler()` and `newTransformerHandler()` methods are supported and whether it is safe to cast your `TransformerFactory` object to a `SAXTransformerFactory`. Use the `FEATURE_XMLFILTER` constant with `getFeature()` to determine if the `newXMLFilter()` methods are also supported.

`newTemplatesHandler()` returns a `TemplatesHandler` object that you can use as an `org.xml.sax.ContentHandler` object to receive SAX events generated by a SAX parser and transform those events into a `Templates` object.

The `newTransformerHandler()` methods are similar: they return a `TransformerHandler` object that can receive SAX events and representing a source document and transform them into a `Result` document. The no-argument version of `newTransformerHandler()` creates a `TransformerHandler` that simply modifies the form of the document without applying a stylesheet to its content. The other two versions of `newTransformerHandler()` use a stylesheet specified either as a `Source` or `Templates` object.

The `newXMLFilter()` methods, if supported, return an `org.xml.sax.XMLFilter` object that can acts as both a sink and a source of SAX events and filters those events by applying the transformation instructions specified by the `Templates` or `Source` objects.

Object	TransformerFactory	SAXTransformerFactory
--------	--------------------	-----------------------

```
public abstract class SAXTransformerFactory extends javax.xml.transform.TransformerFactory {
// Protected Constructors
protected SAXTransformerFactory();
// Public Constants
public static final String           = [quot ] http://javax.xml.transform.sax.SAXTransformerFactory/feature [quot ]
FEATURE;
public static final String           = [quot ] http://javax.xml.transform.sax.SAXTransformerFactory/feature/xmlfilter [quot ]
FEATURE_XMLFILTER;
// Public Instance Methods
public abstract TemplatesHandler newTemplatesHandler() throws
    javax.xml.transform.TransformerConfigurationException;
```

```

public abstract TransformerHandler newTransformerHandler() throws
    javax.xml.transform.TransformerConfigurationException;
public abstract TransformerHandler newTransformerHandler(javax.xml.transform.Source src)
    throws javax.xml.transform.TransformerConfigurationException;
public abstract TransformerHandler newTransformerHandler(javax.xml.transform.Templates templates)
    throws javax.xml.transform.TransformerConfigurationException;
public abstract org.xml.sax.XMLFilter newXMLFilter(javax.xml.transform.Source src)
    throws javax.xml.transform.TransformerConfigurationException;
public abstract org.xml.sax.XMLFilter newXMLFilter(javax.xml.transform.Templates templates)
    throws javax.xml.transform.TransformerConfigurationException;
}

```

TemplatesHandler

Java 1.4

javax.xml.transform.sax

This interface extends `org.xml.sax.ContentHandler` and adds a `getTemplates()` method. An object that implements this interface can be used to receive method calls from some source of SAX events and process those events (as an XSL stylesheet) into a `Templates` object. Obtain a `TemplatesHandler` from a `SAXTransformerFactory`. Register it with the `setContentHandler()` method of an `org.xml.sax.XMLReader` and invoke the `parse()` method of the reader. When `parse()` returns, call the `getTemplates()` method to obtain the `Templates` object.

ContentHandler · TemplatesHandler

```

public interface TemplatesHandler extends org.xml.sax.ContentHandler {
    // Public Instance Methods
    public abstract String getSystemId();
    public abstract javax.xml.transform.Templates getTemplates();
    public abstract void setSystemId(String systemID);
}

```

Returned By: `SAXTransformerFactory.newTemplatesHandler()`

TransformerHandler

Java 1.4

javax.xml.transform.sax

This interface extends `org.xml.sax.ContentHandler` and related interfaces so that it can consume SAX events generated by a `org.xml.sax.SAXReader` or `org.xml.sax.XMLFilter`. Create a `TransformerHandler` by calling one of the `newTransformerHandler()` methods of a `SAXTransformerFactory`.

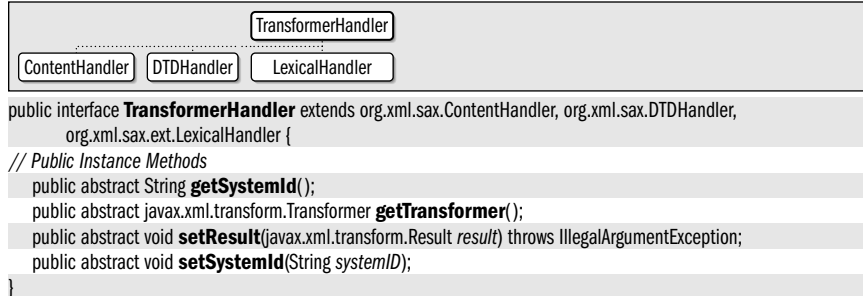
Next, call the `setResult()` method to specify a `Result` object that describes the result document you'd like the transformation to produce. You may also call `getTransformer()` to get the `Transformer` object associated with this `TransformerHandler` if you need to set output properties or parameter values for the transformation.

Now, register the `TransformerHandler` with the `XMLReader` or `XMLFilter` object by calling `setContentHandler()`, `setDTDHandler()`, and `setProperty()`. Use the property name `"http://www.xml.org/sax/properties/lexical-handler"` in the call to `setProperties()` to register the `TransformerHandler` as a `org.xml.sax.ext.LexicalHandler` for the parser or filter.

Finally, invoke one of the `parse()` methods on your `XMLReader` or `XMLFilter` object. This will cause the reader or filter to start parsing the source document and translating it into method calls on the `TransformerHandler`. The `TransformerHandler` will transform those calls as specified in the `Templates` or `Source` object (if any) that was passed to the original call to

TransformerHandler

`newTransformerHandler()` and generate a result document as directed by the `Result` object that was passed to `setResult()`.



Returned By: `SAXTransformerFactory.newTransformerHandler()`

Package javax.xml.transform.stream

Java 1.4

This package contains `Source` and `Result` implementations that work with files and streams.

Classes:

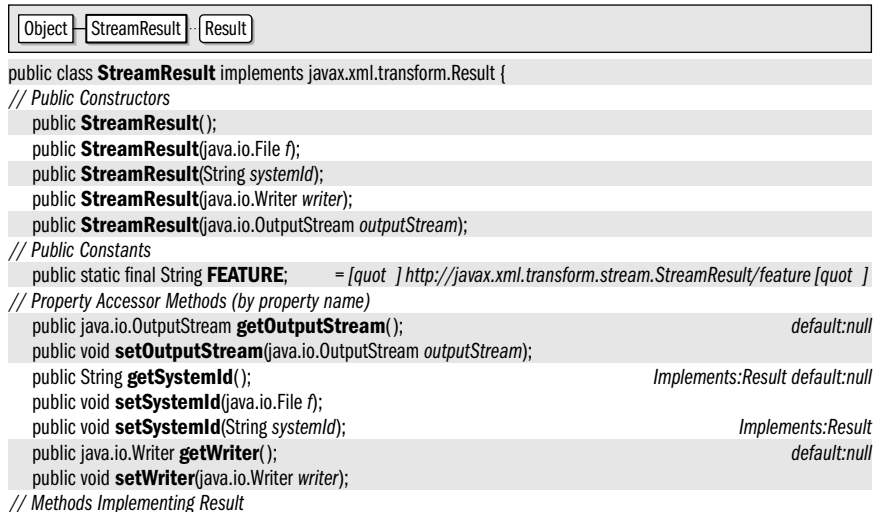
public class **StreamResult** implements `javax.xml.transform.Result`;
public class **StreamSource** implements `javax.xml.transform.Source`;

StreamResult

Java 1.4

`javax.xml.transform.stream`

This class is a `Result` implementation that writes a textual representation of a transformed document to stream or file. Because XML documents define their own encoding, it is usually preferable to construct a `StreamResult` using a `File` or `OutputStream` instead of a character-based `Writer` which may use a different encoding than that specified within the document.



```

    public String getSystemId(); default:null
    public void setSystemId(String systemId);
}

```

StreamSource**Java 1.4****javax.xml.transform.stream**

This class is a **Source** implementation that reads the textual format of an XML document from a file, byte stream, or character stream. Because XML documents declare their own encoding, it is preferable to create a **StreamSource** object from an **InputStream** instead of from a **Reader**, so that the XML processor can correctly handle the declared encoding. When creating a **StreamSource** from a byte stream or character stream, you should provide the system ID (i.e., the filename or URL) by using one of the two-argument constructors or by scaling **setSystemId()**. The system ID is required if the XML file to be processed includes relative URLs to be resolved.

Object	StreamSource	Source
--------	--------------	--------

```

public class StreamSource implements javax.xml.transform.Source {
// Public Constructors
    public StreamSource();
    public StreamSource(java.io.InputStream inputStream);
    public StreamSource(java.io.Reader reader);
    public StreamSource(java.io.File f);
    public StreamSource(String systemId);
    public StreamSource(java.io.Reader reader, String systemId);
    public StreamSource(java.io.InputStream inputStream, String systemId);
// Public Constants
    public static final String FEATURE; = [quot ] http://javax.xml.transform.stream.StreamSource/feature [quot ]
// Property Accessor Methods (by property name)
    public java.io.InputStream getInputStream(); default:null
    public void setInputStream(java.io.InputStream inputStream);
    public String getPublicId(); default:null
    public void setPublicId(String publicId);
    public java.io.Reader getReader(); default:null
    public void setReader(java.io.Reader reader);
    public String getSystemId(); Implements:Source default:null
    public void setSystemId(java.io.File f);
    public void setSystemId(String systemId); Implements:Source
// Methods Implementing Source
    public String getSystemId(); default:null
    public void setSystemId(String systemId);
}

```