intensive. For that reason, the SAX API will tend to be preferred for...is called a "pluggability layer", which allows you to plug into an implementation of the SAX or DOM APIs. The pluggability layer also allows you to plug in an XML processor, letting you control how your XML data is displayed.

The JAXP APIs

The main JAXP APIs are defined in the javax.xml.parsers package. That package contains two vendor-neutral factory classes: SAXParserFactory and DocumentBuilderFactory that give you a SAXParser and a DocumentBuilderFactory respectively. The DocumentBuilderFactory, in turn, creates DOM-compliant Document interface. The factory APIs give you the ability to plug in an XML implementation offered by another vendor without changing your source code. The implementation you get depends on the setting of the javax.xml.parsers.SAXParserFactory and java.xml.parser.DocumentBuilderFactory system properties. The default values (unless overridden at runtime) point to Sun's implementation.

The remainder of this section shows how the different JAXP APIs work when you write an application.

An Overview of the Packages

The SAX and DOM APIs are defined by XML-DEV group and by the W3C, respectively. The libraries that define these APIs are:

- javax.xml.parsers
- org.w3c.dom

The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.

org.xml.sax

Defines the basic SAX APIs.

javax.xml.parsers

The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.

org.w3c.dom

Defines the Document interface (a DOM), as well as classes for all of the components of a DOM.

org.xml.sax

Defines the basic SAX APIs.

javax.xml.parsers

The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.

org.xml.sax

The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.

org.w3c.dom

Defines the Document interface (a DOM), as well as classes for all of the components of a DOM.

org.xml.sax

Defines the basic SAX APIs.

javax.xml.parsers

The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.

org.w3c.dom

Defines the Document interface (a DOM), as well as classes for all of the components of a DOM.

org.xml.sax

Defines the basic SAX APIs.

javax.xml.parsers

The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.

org.w3c.dom

Defines the Document interface (a DOM), as well as classes for all of the components of a DOM.

org.xml.sax

Defines the basic SAX APIs.

javax.xml.parsers

The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.

org.w3c.dom

Defines the Document interface (a DOM), as well as classes for all of the components of a DOM.

org.xml.sax

Defines the basic SAX APIs.

javax.xml.parsers

The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.

org.w3c.dom

Defines the Document interface (a DOM), as well as classes for all of the components of a DOM.

org.xml.sax

Defines the basic SAX APIs.
The public identifier may be specified in addition to the URL. The EntityResolver can then use the public identifier instead of the URL to find the document, for example to access a local copy of the document if one exists.

A typical application implements most of the ContentHandler methods, at a minimum. Since the default implementations of the interfaces ignore all inputs except for fatal errors, a robust implementation may want to implement the ErrorHandler methods, as well.

The SAX Packages

The SAX parser is defined in the following packages listed in Table 4–1.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.xml.sax</td>
<td>Defines the SAX interfaces. The name org.xml is the package prefix that was settled on by the group that defined the SAX API.</td>
</tr>
<tr>
<td>org.xml.sax.ext</td>
<td>Defines SAX extensions that are used when doing more sophisticated SAX processing, for example, to process a document type definition (DTD) or to see the detailed syntax for an element.</td>
</tr>
<tr>
<td>org.xml.sax.helpers</td>
<td>Contains helper classes that make it easier to use SAX—for example, by defining a default handler that has null-methods for all of the interfaces, so you only need to override the ones you actually want to implement.</td>
</tr>
<tr>
<td>javax.xml.parsers</td>
<td>Defines the SAXParserFactory class which returns the SAXParser. Also defines exception classes for reporting errors.</td>
</tr>
</tbody>
</table>

The Document Object Model (DOM) APIs

You use the javax.xml.parsers.DocumentBuilderFactory class to get a DocumentBuilder instance, and use that to produce a Document (a DOM) that conforms to the DOM specification. The builder you get, in fact, is determined by the System property, javax.xml.parsers.DocumentBuilderFactory, which selects the factory implementation that is used to produce the builder. (The platform’s default value can be overridden from the command line.)

You can also use the DocumentBuilderFactory.newInstance() method to create an empty Document that implements the org.w3c.dom.Document interface. Alternatively, you can use one of the builder’s parse methods to create a Document from existing XML data. The result is a DOM tree like that shown in the diagram.

You use the javax.xml.parsers.DocumentBuilderFactory class to get a DocumentBuilder instance, and use that to produce a Document (a DOM) that conforms to the DOM specification. The builder you get, in fact, is determined by the System property, javax.xml.parsers.DocumentBuilderFactory, which selects the factory implementation that is used to produce the builder. (The platform’s default value can be overridden from the command line.)

You can also use the DocumentBuilderFactory.newInstance() method to create an empty Document that implements the org.w3c.dom.Document interface. Alternatively, you can use one of the builder’s parse methods to create a Document from existing XML data. The result is a DOM tree like that shown in the diagram.

Figure 4–2 DOM APIs

You use the javax.xml.parsers.DocumentBuilderFactory class to get a DocumentBuilderFactory instance, and use that to produce a Document (a DOM) that conforms to the DOM specification. The builder you get, in fact, is determined by the System property, javax.xml.parsers.DocumentBuilderFactory, which selects the factory implementation that is used to produce the builder. (The platform’s default value can be overridden from the command line.)

You can also use the DocumentBuilderFactory.newInstance() method to create an empty Document that implements the org.w3c.dom.Document interface. Alternatively, you can use one of the builder’s parse methods to create a Document from existing XML data. The result is a DOM tree like that shown in the diagram.

Note: Although they are called objects, the entries in the DOM tree are actually fairly low-level data structures. For example, under every element node (which corresponds to an XML element) there is a text node which contains the name of the element tag! This issue will be explored at length in the DOM section of the tutorial, but users who are expecting objects are usually surprised to find that invoking the text() method on an element object returns nothing! For a truly object-oriented tree, see the JDOM API at http://www.jdom.org.

The DOM Packages

The Document Object Model implementation is defined in the packages listed in Table 4–2.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.w3c.dom</td>
<td>Defines the DOM programming interfaces for XML, and, optionally, HTML, as specified by the W3C.</td>
</tr>
<tr>
<td>javax.xml.parsers</td>
<td>Defines the DocumentBuilderFactory class and the DocumentBuilder class, which create objects that implement the W3C Document interface. The factory that is used to create the builder is determined by the javax.xml.parsers.DocumentBuilderFactory, which can be set to the default factory or overridden when invoking the new instance method. This package also defines the Document class for creating DOM documents.</td>
</tr>
</tbody>
</table>

Figure 4–2 shows the JAXP APIs in action:
Figure 4–3 shows the XSLT APIs in action.

A TransformerFactory object is instantiated, and used to create a Transformer. The source object is the input to the transformation process. A source object can be created from SAX reader, from a DOM, or from an input stream. Similarly, the result object is the result of the transformation process. That object can be a SAX event handler, a DOM, or an output stream.

When the transformer is created, it may be created from a set of transformation instructions, in which case the specified transformations are carried out. If it is created without any specific instructions, then the transformer object simply copies the source to the result.

### The XSLT Packages

The XSLT APIs are defined in the following packages:

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.transform</td>
<td>Defines the TransformerFactory and Transformer classes, which you use to get an object capable of doing transformations. After creating a transformer object, you call the transform() method, providing it with an input (source) and output (result).</td>
</tr>
<tr>
<td>javax.xml.transform.dom</td>
<td>Classes to create input (source) and output (result) objects from a DOM.</td>
</tr>
<tr>
<td>javax.xml.transform.sax</td>
<td>Classes to create input (source) from a SAX parser and output (result) objects from a SAX event handler.</td>
</tr>
<tr>
<td>javax.xml.transform.stream</td>
<td>Classes to create input (source) and output (result) objects from an I/O stream.</td>
</tr>
</tbody>
</table>

### Compiling and Running the Programs

In the J2EE 1.4 Application Server, the JAXP libraries are distributed in the directory `<J2EE_HOME>/lib/endorsed`. To run the sample programs, you’ll need to use the Java 2 platform’s “endorsed standards” mechanism to access those libraries. For details, see Compiling and Running the Program (page 139).

### Where Do You Go from Here?

At this point, you have enough information to begin picking your own way through the JAXP libraries. Your next step from here depends on what you want to accomplish. You might want to go to:

- Simple API for XML (page 127) if the data structures have already been determined, and you are writing a server application or an XML filter that needs to do fast processing.
- Document Object Model (page 187) if you need to build an object tree from XML data so you can manipulate it in an application, or convert an in-memory tree of objects to XML. This part of the tutorial ends with a section on namespaces.
- XML Stylesheet Language for Transformations (page 261) if you need to transform XML tags into some other form, or if you want to generate XML output, or (in combination with the SAX API) if you want to convert legacy data structures to XML.