# BEA WebLogic Server

## Programming WebLogic XML

BEA WebLogic Server 6.0
Document Date: May 15, 2001

## Copyright



## Restricted Rights Legend



## Trademarks or Service Marks



**Programming WebLogic XML**

| Part Number | Document Date | Software Version |
|---|---|---|
| N/A | May 15, 2001 | BEA WebLogic Server Version 6.0 |

# Contents

## About This Document



## 1. XML Overview

## 2. Developing XML Applications with WebLogic Server



## 3. XML Administration

## 4. XML Reference



## Index

# About This Document

This document explains how to use the BEA WebLogic Server™ XML software. It defines concepts associated with using the XML software and describes the development process for XML applications. In addition, the document includes descriptions of the application programming interfaces (APIs), administrative tasks, and XML tools.

The document is organized as follows:

- Chapter 1, "XML Overview," provides a basic description of the XML software and its components.
- Chapter 2, "Developing XML Applications with WebLogic Server," describes how to develop XML applications using WebLogic Server and XML tools.
- Chapter 3, "XML Administration," describes the Administration Console XML Registry and how to perform XML configuration tasks.
- Chapter 4, "XML Reference," provides pointers to specifications and application programming interfaces supported by the XML software.

## Audience

This document is written for system administrators and programmers who design, develop, configure, and manage XML applications. It is assumed that readers know Web technologies, XML, XSLT, the Java programming language, and the Servlet and JSP APIs of the J2EE specification.

# e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the WebLogic Server Product Documentation page at http://e-docs.bea.com/wls/docs60.

# How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at http://www.adobe.com.

# Related Information

For related information about XML, see “Learning About XML” on page 1-2.

# Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

# Documentation Conventions

The following documentation conventions are used throughout this document.

| Convention | Usage |
|---|---|
| Ctrl+Tab | Keys you press simultaneously. |
| *italics* | Emphasis and book titles. |

| Convention | Usage |
|---|---|
| `monospace text` | Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard.<br>*Examples*:<br>`import java.util.Enumeration;`<br>`chmod u+w *`<br>`config/examples/applications`<br>`.java`<br>`config.xml`<br>`float` |
| *`monospace italic text`* | Variables in code.<br>*Example*:<br>`String CustomerName;` |
| UPPERCASE TEXT | Device names, environment variables, and logical operators.<br>*Examples*:<br>LPT1<br>BEA_HOME<br>OR |
| `{ }` | A set of choices in a syntax line. |
| `[ ]` | Optional items in a syntax line. *Example*:<br>`java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]` |
| `\|` | Separates mutually exclusive choices in a syntax line. *Example*:<br>`java weblogic.deploy [list\|deploy\|undeploy\|update] password {application} {source}` |
| `...` | Indicates one of the following in a command line:<br>■ An argument can be repeated several times in the command line.<br>■ The statement omits additional optional arguments.<br>■ You can enter additional parameters, values, or other information |

| Convention | Usage |
|---|---|
| .<br>.<br>. | Indicates the omission of items from a code example or from a syntax line. |

CHAPTER

# 1 XML Overview

The following sections provide an overview of XML technology and WebLogic Server:

- What Is XML?
- What Are XSL and XSLT?
- Common Uses of XML and XSLT
- WebLogic Server XML Features
- WebLogic Server XML Components
- XML Module Components

## What Is XML?

Extensible Markup Language (XML) is a markup language for documents containing structured information. It is a simplified version of Standard Generalized Markup Language (SGML). XML has become an industry standard for delivering content on the Internet. Because it provides a facility to define new tags, XML is also extensible.

Like HTML, XML uses tags to describe content. However, rather than focusing on the presentation of content, the tags in XML describe the meaning and hierarchical structure of data. This functionality allows for the sophisticated data types that are required for efficient data interchange between different programs and systems. Further, because XML enables separation of content and presentation, the content, or data, is portable across heterogeneous systems.

The XML syntax uses matching start and end tags to mark up information. Information delimited by tags is called an element. Elements can also have attributes that are defined in the form of name-value pairs.

The syntactic meaning associated with the tags of an XML document is defined in a Document Type Definition (DTD). A DTD describes the elements and attributes that are valid in an XML document, and the contexts in which they are valid. In other words, a DTD specifies which tags are allowed within certain other tags, and which tags and attributes are optional.

Schemas are a recent development in XML specifications and are intended to supersede DTDs. They describe XML documents with more flexibility and detail than DTDs do. The schema specification, currently under development, is a product of the World Wide Web Consortium (W3C) and is intended to address many limitations of DTDs.

**Note:** In this version of WebLogic Server, the final version of the schema specification is not supported. Therefore, the schema support in this version should be considered experimental; it should not be used in a production environment.

# Why Use XML?

An industry typically uses data exchange methods that are meaningful and specific to that industry. With the advent of e-commerce, businesses conduct an increasing number of relationships with a variety of industries and, therefore, must develop expert knowledge of the various protocols used by those industries for electronic communication.

The extensibility of XML makes it a very effective tool for standardizing the format of data interchange among various industries. For example, when message brokers and workflow engines must coordinate transactions among multiple industries or departments within an enterprise, they can use XML to combine data from disparate sources into a format that is understandable by all parties.

# Learning About XML

To learn about XML, see the following online courses and tutorials:

- A Technical Introduction to XML at "http://www.xml.com/pub/a/98/10/guide0.html"
- XML Authoring Tutorial at "http://www.xml.com/pub/r/32."
- Working with XML and Java at "http://java.sun.com/xml/tutorial_intro.html."
- Tutorials for using the Java 2 platform and XML technology at "http://developerlife.com/."
- XML, Java, and the Future of the Web at "http://www.xml.com/pub/a/w3j/s3.bosak.html."
- Chapter 14 of The XML Bible: XSL Transformations at "http://metalab.unc.edu/xml/books/bible/updates/14.html."
- XSL Tutorial by Miloslav Nic at http://zvon.vscht.cz/HTMLonly/XSLTutorial/Books/Book1/index.html.

# What Are XSL and XSLT?

The Extensible Stylesheet Language (XSL) is a W3C standard for describing presentation rules that apply to XML documents. XSL includes both a transformation language, (XSLT), and a formatting language. These two languages function independently of each other. XSLT is an XML-based language and W3C specification that describes how to transform an XML document into another XML document, or into HTML, PDF, or some other document definition format.

An XSLT processor accepts as input an XML document and an XSLT document. The template rules contained in an XSLT document include patterns that specify the XML tree to which the rule applies. The XSLT processor scans the XML document for patterns that match the rule, and then it applies the template to the appropriate section of the original XML document.

# Common Uses of XML and XSLT

How you use XML and XSLT depends on your particular business needs.

## Using XML and XSLT to Separate Content from Presentation

XML and XSLT are often used in applications that support multiple client types. For example, suppose you have a Web-based application that supports both browser-based clients and Wireless Application Protocol (WAP) clients. These clients understand different markup languages, HTML and Wireless Markup Language (WML), respectively, but your application must deliver content that is appropriate for both.

To accomplish this goal, you can write your application to first produce an XML document that represents the data it is sending to the client. Then the application can transform the XML document that represents the data into HTML or WML, depending on the client's browser type. Your application can determine the client browser type by examining the `User-Agent` request header of an HTTP request. Once the application knows the client browser type, it uses the appropriate XSLT style sheet to transform the document into the correct markup language. See the SnoopServlet example included in the `examples/servlets` directory of your WebLogic Server distribution for an example of how to access this type of header information.

This method of rendering the same XML document using different markup languages in respective client types helps concentrate the effort required to support multiple client types into the development of the appropriate XSLT style sheets. Additionally, it allows your application to adapt to other clients types easily, if necessary.

For additional information about XSLT, see "Additional Resources" on page 4-5.

## XML as a Message Format for Business-to-Business Communication

In a business-to-business (B2B) environment, Company A and Company B want to exchange information about e-commerce transactions in which both are involved. Company A is a major e-commerce site. Company B is a small affiliate that sells Company A's products to a niche group of customers. When Company B sends customers to Company A, Company B is compensated in two ways: it receives, from Company A, both money and information about other customers that make the same sort of purchases as those made by the customers referred by Company B. To exchange information, Company A and Company B must agree on a data format for information that is machine readable and that operates with systems from both companies easily.

XML is the logical data format to use in this scenario, but selecting this format is only the first step. The companies must then agree on the format of the XML messages to be exchanged. Because Company A has a one-to-many relationship with its affiliates, Company A must define the format of the XML messages they will send and accept.

To define the format of XML messages, or XML documents, Company A creates two document type definitions (DTDs): one that describes the information that A will provide about customers and one that describes the information that A wants to receive about a newly affiliated company. Company B must also create two DTDs: one to process the XML documents received from Company A and one to prepare an XML document in a format that can be processed by Company A.

# WebLogic Server XML Features

WebLogic Server consolidates XML technologies applicable to WebLogic Server and XML applications based on WebLogic Server. The XML Module is a set of XML components that is installed and integrated automatically when you install WebLogic Server. These components facilitate the development and use of XML applications. Using XML Module and WebLogic Server, customers can build XML applications that use standard parsers, custom-generated parsers, XSLT transformers, and DTDs to process and convert XML files.

# Parsing XML Documents

WebLogic Server includes a built-in parser that is based on the Apache Xerces parser version 1.2.0. You can use the built-in parser in either Simple API For XML (SAX) mode or Document Object Model (DOM) mode.

You can also use the parser of your choice for selected XML applications or for all your XML applications: custom-generated parsers, different versions of the Apache Xerces parser, the Sun parser, and so on.You can configure a single WebLogic Server to use one parser for a particular application and use another parser for a different application. WebLogic Server makes parser capabilities available through the XML Registry. To use these capabilities, use the Administration Console to configure the appropriate parser classes in the XML Registry. For more information about using the XML Registry to specify different parsers, see "Configuring a Parser Other Than the Built-In Parser" on page 3-5.

# Generating Custom Parsers

Using the WebLogic Parser Generator you can generate a parser that is tailored for a particular DTD. Generated parsers use the SAX API. They are efficient and fast, but should only parse XML documents that use the same DTD used to generate the parser. For more information about the limitations, behaviors, and performance of generated parsers, refer to:

- "Limitations of Generated Parsers" on page 2-16
- "Performance and Usage Considerations for Generated Parsers" on page 2-17

# Validating XML Documents

WebLogic Server supports the use of DTDs to validate XML documents.

As mentioned earlier, an XML DTD contains markup declarations that provide a grammar for a class of documents. An XML document is considered *valid* if 1) it has an associated DTD, and 2) it complies with the constraints expressed in the associated DTD.

Validation is a powerful tool for ensuring that an XML document contains all the information required by your application. An XML parser automatically checks a document to ensure that it is *well-formed*, or that it follows all the rules in the W3C Recommendation for XML 1.0. However, to *validate* a document, you must include in your document either a DTD or a reference to an external DTD. You specify how the DTD will be handled within the document type declaration.

## Transforming XML Documents

WebLogic Server includes a built-in XSLT processor that is based on the Apache Xalan-Java XSLT processor version 1.2. You can use this built-in XSLT processor or other XSLT processors in your XML application to transform XML documents. To do so, you must define the processor of choice in your application code. For more information about transforming XML documents, see "Transforming XML Documents" on page 2-8.

## Resolving External Entities

The XML Module supports external entity resolution through the XML Registry. This feature of the registry eliminates the need to construct and set SAX EntityResolvers. To use this feature, open the Administration Console and use the XML Registry to associate the pathnames of local copies of external entities with their `Public id` and/or `System id`. These entities are then resolved automatically by WebLogic Server at parse time. For more information about using the XML Registry for external entity resolution, see "Using the XML Registry" on page 3-2.

# WebLogic Server XML Components

The WebLogic Server distribution provides XML components packaged as follows:

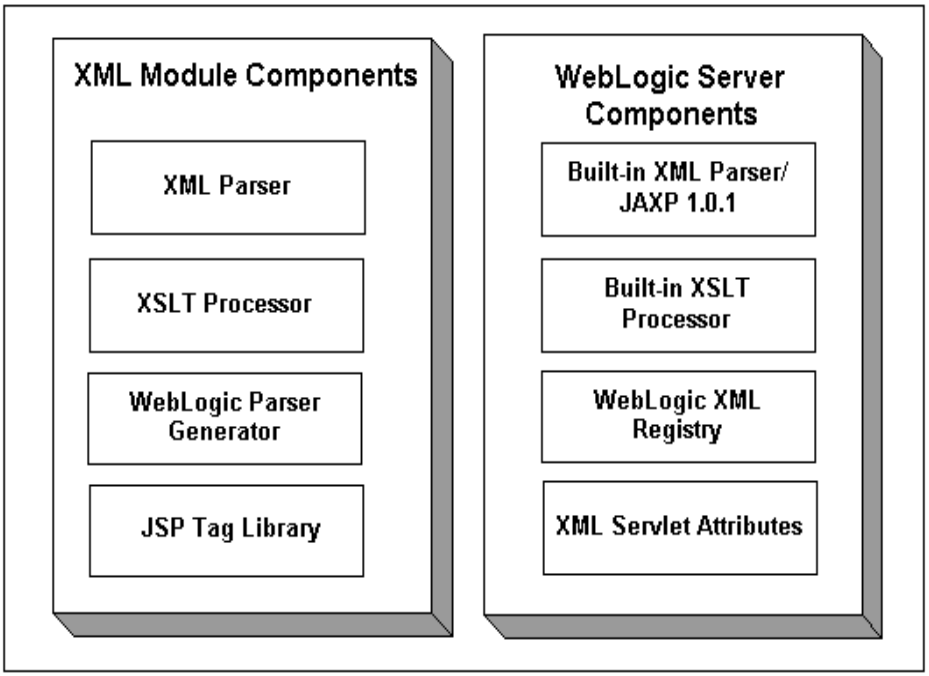
- `weblogic.jar` contains some WebLogic Server XML components.
- `xmlx.jar` and `xmlx-tags.jar` contain the XML Module components.

All of these components are automatically installed when you install the WebLogic Server distribution.

WebLogic Server includes the basic components needed to build and run XML applications:

- Built-in XML Parser
- WebLogic Integrated Java API for XML Parsing (JAXP) 1.0.1
- Built-in XSLT Processor
- WebLogic XML Registry
- XML Servlet Attributes

**Figure 1-1 WebLogic Server XML Components**



# Built-in XML Parser

In WebLogic Server Version 6.0, the built-in parser is based on the Apache Xerces parser 1.2.0. This parser replaces the Sun parser that shipped with WebLogic Server 5.1. By default, the built-in parser is used to parse XML documents. Because the built-in parser is in the `weblogic.apache.xerces` package, it does not interfere with any other version of the Xerces parser that may be defined in the WebLogic Server classpath. If you want to use a different parser, use the WebLogic XML Registry to configure the parser of your choice as the default parser. This design minimizes the work required to write, build, and run XML applications on WebLogic Server that use different parsers.

**Note:** The built-in parser is in the `weblogic.Apache.Xerces` package.

# WebLogic Integrated Java API for XML Parsing (JAXP) 1.0.1

Java API for XML Parsing (JAXP) 1.0.1 is a Java-standard, parser-independent API for XML. When you use JAXP, you can also use the WebLogic XML Registry to control which parsers are used without having to write application code that is dependent on a particular parser. The use of the XML Registry to configure parsers supersedes the use of the JAXP parser factory system properties, which is the default JAXP mechanism for configuring a parser.

JAXP 1.0.1 supports two modes for processing XML documents: SAX 1.0 and DOM Level 1.

- For SAX processing, JAXP defines a factory API that enables applications to configure and obtain a SAX-based parser to parse XML documents.
- For DOM processing, JAXP defines a factory API that enables applications to configure and obtain a parser to parse XML documents into a DOM document tree.

# Built-in XSLT Processor

WebLogic Server includes an XSLT processor, which is based on the Apache Xalan-Java XSLT processor 1.2, as the built-in XSLT processor. You can use the Xalan-based processor to transform XML documents into HTML, WML, text, or other XML document types. XSLT transformations are not part of JAXP 1.0.1, so you must use Xalan specific code in your XML application to instantiate and run XSLT transformation using the built-in XSLT processor. Because the Xalan processor classes are provided in the `weblogic.apache.xalan` package, the built-in XSLT processor does not interfere with any other version of the Xalan XSLT processor that may be defined in the WebLogic Server classpath.

# WebLogic XML Registry

The XML Registry simplifies XML administration and configuration tasks by separating these tasks from the XML application. Use the Administration Console (a graphical user interface, or GUI, for WebLogic Server administration) to configure and assign XML Registries.

**Note:** Each WebLogic Server domain can include any number of registries; each WebLogic Server in a domain can be assigned zero or one registry.

The advantages of the XML Registry are as follows:

- You can specify the parser at deployment time, not only at build time.
- You do not need to include any parser-dependent code in your applications.
- You can support multiple parsers in a single server more conveniently.

You can use the XML registry to perform the following tasks:

- Specify an alternative default XML parser instead of the built-in parser shipped in this version of WebLogic Server.
- Specify an XML parser that should be used to process a particular document type (including custom-generated parsers).
- Specify external entities that are to be resolved locally. Then, you can store local copies of external entities in the Administration Server's file system so that they

can be distributed automatically to the server's parser at parse time. This feature eliminates the need to construct and set SAX EntityResolvers and the need for the parser to consult an external Web site, thus improving parser performance.

All the preceding capabilities are available if your application uses the standard Java API for XML Parsing (JAXP), which is included in this version of WebLogic Server. These capabilities are for use on the server side only.

## XML Servlet Attributes

WebLogic Server supports two special XML servlet attributes: `org.xml.sax.HandlerBase` and `org.w3c.dom.Document`. These attributes:

- Provide access to XML from servlets that use SAX or DOM parsers. For SAX, you use the `setAttribute` method to access the `org.xml.sax.HandlerBase` attribute. For DOM, you use the `getAttribute` method to access the `org.w3c.dom.Document` attribute.
- Work with parser and entity resolver configuration information in the XML Registry.

**Note:** The `setAttribute` and `getAttribute` methods are provided for convenience only; they are not required to parse XML from a servlet.

# XML Module Components

The XML Module is a set of XML components that is installed and integrated automatically when you install the WebLogic Server software (see Figure 1-1). These components facilitate the development and use of XML applications. Specifically, the module components enable the use of the parser generator and the JSP tag library. Although the module components do not make up a separate software package, they perform a discrete set of functions, and are therefore referred to collectively as a module.

**Note:** The XML Module can be updated and released separately from WebLogic Server.

The XML Module contains the following components:

- XML Parser
- XSLT Processor
- WebLogic Parser Generator
- WebLogic XSLT JSP Tag Library

# XML Parser

The XML Module includes the Apache Xerces parser version 1.2.0. In this version of WebLogic Server, this component is functionally the same as the WebLogic Server built-in XML parser.

# XSLT Processor

The XML Module includes Apache Xalan-Java XSLT processor 1.2. In this version of WebLogic Server, this component is functionally the same as the WebLogic Server built-in XSLT processor.

# WebLogic Parser Generator

The parser generator tool is built by BEA to achieve custom parsing performance while using standard XML APIs. This tool enables you to generate customized SAX parsers that are specific to the document types represented by DTDs. Because each generated parser is built for a specific DTD, it cannot be used to parse XML documents written for other DTDs.

The parser generator is included in `xmlx.jar`, which is installed when you install your WebLogic Server distribution.

Using the parser generator has advantages and disadvantages. The advantages are better performance while using standard JAXP APIs, a smaller code footprint, and less server resource utilization. The disadvantages are that the parser generator produces a SAX 1.0 parser (instead of SAX 2.0); it is neither a fully validating parser nor a non-validating parser; and it does not support DOM.

After you use this tool to generate a custom parser, you can configure the XML registry in WebLogic Server so that the custom-generated parser is automatically used to parse a particular XML document.

# WebLogic XSLT JSP Tag Library

The JSP tag library provides a simple tag that enables access to the built-in XSLT processor from within a Java Server Pages (JSP) running on WebLogic Server. Currently, this tag supports the built-in XSLT processor only; you cannot use the tag to parse an XML document from within a JSP using a different parser.

The JSP tag library is included in `xmlx-tags.jar`, which is installed when you install your WebLogic Server distribution.

**Note:** The JSP tag library is provided for convenience only; it is not required to access XSLT processors from within a JSP.

CHAPTER

# 2 Developing XML Applications with WebLogic Server

The following sections describe how to use the Java programming language and WebLogic Server to develop XML applications. It is assumed that you know how to use Java Servlets and Java Server Pages (JSPs) to write Java applications. For information about how to write servlet and JSP applications, see *Programming WebLogic HTTP Servlets* at http://e-docs.bea.com/wls/docs60/servlet/index.html and *Programming WebLogic JSP* at http://e-docs.bea.com/wls/docs60/jsp/index.html.

- Using WebLogic Server Features For XML Application Development
- Generating XML
- Transforming XML Documents
- Using Parsers Other Than the Built-In Parser

## Using WebLogic Server Features For XML Application Development

To facilitate XML application development and the work required to move XML applications built on WebLogic Server to other Web application servers, WebLogic Server implements the Java API for XML Parsing (JAXP). JAXP was developed by

Sun Microsystems to make XML applications portable; it provides basic support for parsing and manipulating XML documents through a standardized set of Java platform APIs. JAXP 1.0.1, included in the WebLogic Server distribution, is configured to use the built-in parser. Therefore, by default, XML applications built using WebLogic Server use JAXP.

The WebLogic Server distribution contains the interfaces and classes needed for JAXP 1.0.1. JAXP 1.0.1 contains explicit support for SAX 1.0 and DOM Level 1. The Javadoc for JAXP is included with the WebLogic Server online reference documentation. The WebLogic Server distribution does not include the Sun Projectx parser.

# Advantage of Using JAXP

When you write XML applications, the application must get the XML document and parse it to make its contents available to the application. To accomplish this task, you can use either the SAX or the DOM API. Both APIs provide extensive functionality and are very useful. However, to use either API, you must import and reference a parser class from a parser vendor's Java code. The problem with this approach is twofold:

- The use of SAX or DOM requires knowledge of the specific implementation of the vendor's parser.
- To make a change to the parser class used, you must change the application code and recompile it.

These problems could be avoided if the XML parser were completely pluggable. The Java API for XML Parsing (JAXP) provides a parser Pluggability Layer. The JAXP Pluggability Layer allows a compliant SAX or DOM parser to be accessed through factory classes in the `javax.xml.parsers` package.

# Using JAXP with SAX

The following code example shows how to provide an implementation of the `org.xml.sax.HandlerBase` class to a `SAXParser` implementation and then parse an XML document:

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

...
MyHandler handler = new MyHandler();
// MyHandler extends org.xml.sax.HandlerBase.

  //Obtain an instance of SAXParserFactory.
  SAXParserFactory spf = SAXParserFactory.newInstance();
  //Specify a validating parser.
  spf.setValidating(true); // Requires loading the DTD.
  //Obtain an instance of a SAX parser from the factory.
  SAXParser sp = spf.newSAXParser();
  //Parse the documnt.
  sp.parse(inputFile, handler);
...
```

**Note:** If you want to use a parser other than the default, built-in parser, you must use the WebLogic Server Administration Console to specify the parser in the XML Registry; otherwise the `newInstance` method returns the built-in parser. For instructions about configuring WebLogic Server to use a parser other than the built-in parser, see "XML Registry Configuration Tasks" on page 3-4.

# Using JAXP and DOM

The following code example shows how to provide an implementation of the `org.w3c.dom.Document` class to a `DocumentBuilder` implementation and then parse an XML document:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;

...
//Obtain an instance of DocumentBuilderFactory.
DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
//Specify a validating parser.
dbf.setValidating(true); // Requires loading the DTD.
//Obtain an instance of a DocumentBuilder from the factory.
DocumentBuilder db = dbf.newDocumentBuilder();
//Parse the document.
Document doc = db.parse(inputFile);
...
```

**Note:** If you want to use a parser other than the built-in parser, you must use the WebLogic Server Administration Console to specify the parser in the XML Registry; otherwise the `newInstance` method returns the built-in parser. For instructions about configuring WebLogic Server to use a parser other than the built-in parser, see "XML Registry Configuration Tasks" on page 3-4.

# XML Registry Configuration Options

As mentioned previously, you use the Administration Console XML Registry to configure the following:

- Server-wide default parser
- Per-doctype parsers, which supersede the default parser for the specified doctype
- External entity resolution, or the process that an XML parser goes through when requested to find an external file in the course of parsing an XML document

When you make configuration changes using the XML Registry, the changes take effect immediately, that is, it is not necessary to recompile the XML application code or to restart the server. You can make changes at deploy time or run time without modifying and recompiling the XML application code or restarting the server.

# Servlet Programming Features

Support for the `setAttribute` and `getAttribute` methods was added to version 2.2 of the Java Servlet Specification. Attributes are objects associated with a request. The request object encapsulates all information from the client request. In the HTTP protocol, this information is transmitted from the client to the server by the HTTP headers and message body of the request.

With WebLogic Server, you can use these methods to parse XML documents. The `setAttribute` method is used for SAX mode parsing; the `getAttribute` method, for DOM mode parsing.

## Using the org.xml.sax.HandlerBase Attribute to Parse a Document

The following code example shows how to use the `setAttribute` method:

```
import weblogic.servlet.XMLProcessingException;
...
public final void doPost(HttpServletRequest request,
                         HttpServletResponse response)
   throws ServletException, IOException  {
   try {// MyHandler extends org.xml.sax.HandlerBase.
      request.setAttribute(
              "org.xml.sax.HandlerBase", new MyHandler());
    } catch(XMLProcessingException xpe) {
      System.out.println("Error in processing XML");
      xpe.printStackTrace();
      return;
    }
...
```

**Note:** This code example shows a simple way to parse a document using SAX and the `setAttribute` method. This is a WebLogic Server convenience feature, and it is not supported by other servlet vendors. Therefore, if you plan to run your application on other servlet platforms, do not use this feature.

## Using the org.w3c.dom.Document Attribute to Parse a Document

The following code example shows how to use the `getAttribute` method.

```
import org.w3c.dom.Document;
import weblogic.servlet.XMLProcessingException;

...

public final void doPost(HttpServletRequest request,
                         HttpServletResponse response)
throws ServletException, IOException  {

try {
   Document doc = request.getAttribute("org.w3c.dom.Document");
 } catch(XMLProcessingException xpe) {
   System.out.println("Error in processing XML");
   xpe.printStackTrace();
   return;
 }
...
```

**Note:** This code example shows a simple way to parse a document using DOM and the `getAttribute` method. This is a WebLogic Server convenience feature, and it is not supported by other servlet vendors. Therefore, if you plan to run your application on other servlet platforms, do not use this feature.

# Generating XML

This section describes how to generate XML documents from a DOM document tree and by using JSP.

## Generating XML from a DOM Document Tree

To generate an XML document from a DOM document tree, you can use the class `weblogic.apache.xml.serialize`. You can use this class to convert a DOM document tree to XML text. For a description of this class, see Javadoc for `weblogic.apache.xml.serialize`.

The following code example shows how to used this class.

```
package test;

import java.io.OutputStreamWriter;
import java.util.Date;
import java.text.DateFormat;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import weblogic.apache.xerces.dom.DocumentImpl;
import weblogic.apache.xml.serialize.DOMSerializer;
import weblogic.apache.xml.serialize.XMLSerializer;

public class WriteXML {

  public static void main(String[] args) throws Exception {

    // Create a DOM tree.
    Document doc= new DocumentImpl();
    Element message = doc.createElement("message");
    doc.appendChild(message);
    Element text = doc.createElement("text");
```

```
    text.appendChild(doc.createTextNode("Hello world."));
    message.appendChild(text);
    Element timestamp = doc.createElement("timestamp");
    timestamp.appendChild(
      doc.createTextNode(
               DateFormat.getDateInstance().format(new Date()))
    );
    message.appendChild(timestamp);

    // Serialize the DOM to XML text and output to stdout.
    DOMSerializer xmlSer =
     new XMLSerializer(new OutputStreamWriter(System.out),null);
    xmlSer.serialize(doc);
  }
}
```

# Generating XML Documents by Using JSP

The most common reason for using JSP is to generate HTML, but JSP can also be used to generate an XML document.

The main requirement for using JSP to generate XML is that the JSP page set the content type of the page as follows:

```
<%@ page contentType="text/xml"%>
    ... XML document
```

The following code shows an example of how to use JSP to generate an XML document:

```
<?xml version="1.0">

<%@ page contentType="text/xml"
import="java.text.DateFormat,java.util.Date" %>

<message>
  <text>
    Hello World.
  </text>
  <timestamp>
<%
out.print(DateFormat.getDateInstance().format(new Date()));
%>
  </timestamp>
</message>
```

For more information about using JSP to generate XML, see http://java.sun.com/products/jsp/html/JSPXML.html.

# Transforming XML Documents

WebLogic Server includes a built-in XSLT processor, which is based on the Apache Xalan XSLT processor. The built-in XSLT processor uses the built-in parser by default. JAXP does not provide a standard API to access the built-in XSLT processor, so the XSLT processor must be handled in your application programmatically.

In an XSLT transformation, an XSLT processor reads both an XML document and an XSLT style sheet. Based on the instructions in the XSLT style sheet, the transformer outputs a new XML document, an HTML document, a WML document, and so on.

WebLogic Server supports two ways of using the XSLT processor:

- You can use the standard Xalan API to instantiate and invoke the built-in XSLT processor from the XML application.
- You can use the JSP tag provided by WebLogic Server to invoke the built-in XSLT processor from a JSP. The JSP tag `x:xslt` is a convenience method that is provided in a JSP tag library.

## Using the XLST Processor to Transform XML Documents

The following code example shows how to construct a simple DOM tree (for the same XML document used in `HelloWorld.jsp`) and then serialize it to XML text. In the example, the XML text is printed to the standard output using the `System.out` method.

**Note:** The following code example was taken from the Apache Xalan samples, which are available at http://xml.apache.org.

```
*
* This code example was taken from code examples provided by the
* Apache Software Foundation. It consists of voluntary
* contributions made by many individuals on behalf of the Apache
* Software Foundation and was originally based on software
```

```
 * copyright (c) 1999, Lotus Development Corporation.,
 * http://www.lotus.com. For more information on the Apache
 * Software Foundation, please see <http://www.apache.org/>.
 */

import org.xml.sax.SAXException;
import org.apache.xalan.xslt.XSLTProcessorFactory;
import org.apache.xalan.xslt.XSLTInputSource;
import org.apache.xalan.xslt.XSLTResultTarget;
import org.apache.xalan.xslt.XSLTProcessor;

/**
 * Simple sample code to show how to run the XSL processor
 * from the API.
 */
public class SimpleTransform
{
        public static void main(String[] args)
    throws java.io.IOException,
           java.net.MalformedURLException,
           org.xml.sax.SAXException
        {
    // Have the XSLTProcessorFactory obtain a interface to a
    // new XSLTProcessor object.
    XSLTProcessor processor =
XSLTProcessorFactory.getProcessor();

    // Have the XSLTProcessor processor object transform
"foo.xml"
    // to System.out, using the XSLT instructions found in
"foo.xsl".

    processor.process(new XSLTInputSource("foo.xml"),
                      new XSLTInputSource("foo.xsl"),
                      new XSLTResultTarget(System.out));
        }
}
```

For additional information about the Xalan samples, see the Apache Xalan Javadoc "at http://e-docs.bea.com/wls/docs60/Xalan/index.html" and the *Overview* and *Getting Started* documents on the Apache Web site at `http://xml.apache.org/xalan`.

# Using the JSP Tag to Transform XML Documents

WebLogic Server provides a small JSP tag library for convenient access to an XSLT processor from within a JSP. You can use this tag to transform XML documents, but it is not required.

The JSP tag library consists of one main tag, `x:xslt`, and two subtags you can use within the `x:xslt` tag: `x:stylesheet` and `x:xml`.

## XSLT JSP Tag Syntax

The XSLT JSP tag syntax is based on XML. A JSP tag consists of a start tag, an optional body, and a matching end tag. The start tag includes the element name and optional attributes.

The following syntax describes how to use the three XSLT JSP tags provided by WebLogic Server in a JSP. The attributes are optional, as are the subtags `x:stylesheet` and `x:xml`. The tables following the syntax describe the attributes of the `x:xslt` and `x:stylesheet` tags; the `x:xml` tag does not have any attributes.

```
<x:xslt  [xml="uri of XML file"]
         [media="media type to determine stylesheet"]
         [stylesheet="uri of stylesheet"]
    <x:xml>In-line XML goes here
    </x:xml>
    <x:stylesheet [media="media type to determine stylesheet"]
                  [uri="uri of stylesheet"]
    </x:stylesheet>
</x:xslt>
```

The following table describes the attributes of the `x:xslt` tag.

| x:xslt Tag Attribute | Required | Data type | Description |
|---|---|---|---|
| xml | No | String | Specifies the location of the XML file that you want to transform. The location is relative to the document root of the Web application in which the tag is used. |

| x:xslt Tag Attribute | Required | Data type | Description |
|---|---|---|---|
| media | No | String | Defines the document output type, such as HTML or WML, that determines which stylesheet to use when transforming the XML document.<br>This attribute can be used in conjunction with the `media` attribute of any enclosed `x:stylesheet` tags within the body of the `x:xslt` tag. The value of the `media` attribute of the `x:xslt` tag is compared to the value of the `media` attribute of any enclosed `x:stylesheet` tags. If the values are equal, then the stylesheet specified by the `uri` attribute of the `x:stylesheet` tag is applied to the XML document.<br>**NOTE**: It is an error to set both the `media` and `stylesheet` attributes within the same `x:xslt` tag. |
| stylesheet | No | String | Specifies the location of the stylesheet to use to transform the XML document. The location is relative to the document root of the Web application in which the tag is used.<br>**NOTE**: It is an error to set both the `media` and `stylesheet` attributes within the same `x:xslt` tag. |

The following table describes the attributes of the `x:stylesheet` tag.

| x:stylesheet Tag Attribute | Required | Data type | Description |
|---|---|---|---|
| media | No | String | Defines the document output type, such as HTML or WML, that determines which stylesheet to use when transforming the XML document.<br>Use this attribute in conjunction with the `media` attribute of enveloping x:xslt tag. The value of the `media` attribute of the `x:xslt` tag is compared to the value of the `media` attribute of the enclosed `x:stylesheet` tags. If the values are equal, then the stylesheet specified by the `uri` attribute of the `x:stylesheet` tag is applied to the XML document. |
| uri | No | String | Specifies the location of the stylesheet to use when the value of the `media` attribute matches the value of the `media` attribute of the enveloping `x:xslt` tag. The location is relative to the document root of the Web application in which the tag is used. |

## XSLT JSP Tag Usage

The `x:xslt` tag can be used with or without a body, and its attributes are optional. This section describes the rules that dictate how the tag behaves depending on whether you specify a body or one or more attributes.

If the `x:xslt` JSP tag is an empty tag (no body), the following statements apply:

- If no attributes are set, the XML document is processed using the servlet path and the default media stylesheet. You specify the default media stylesheet in your XML file with the `<?xml-stylesheet>` processing instruction; the default stylesheet is the one that does not have a `media` attribute.

  This type of processing allows you to register the JSP page that contains the tag extension as a file servlet that performs XSLT processing.

- If only the `media` attribute is set, the XML document is processed using the servlet path and the specified media type. The value of the `media` type attribute of the `x:xslt` tag is compared to the value of the `media` attribute of any `<?xml-stylesheet>` processing instructions in your XML document; if any match then the corresponding stylesheet is applied. If none match then the default media stylesheet is used. The media type attribute is used to define the document output type (for example, XML, HTML, postscript, or WML). This feature enables you to organize stylesheets by document output type.

- If only the `xml` attribute is set, the specified XML document is processed using the default media stylesheet.

- If the `media` and `xml` attributes are set, the specified XML document is processed using the specified media type.

- If the `stylesheet` attribute is defined, the XML document is processed using the specified stylesheet.

**Caution:** It is an error to set both the `media` and `stylesheet` attributes.

An XSLT JSP tag that has a body may contain `<x:xml>` tags and/or `<x:stylesheet>` tags.

- The `<x:xml>` tag allows you specify an XML document for inline processing. This tag has no attributes.

- The `<x:stylesheet>` tag, when used without any attributes, allows you specify the default stylesheet inline.
- Use the `uri` attribute of the `<x:stylesheet>` tag to specify the location of the default stylesheet.
- If you want to specify different stylesheets for different media types, you can use multiple `<x:stylesheet>` tags with different values for the `media` attribute. You can specify a stylesheet for each media type in the body of the tag, or specify the location of the stylesheet with the `uri` attribute.

## Transforming XML Documents Using an XSLT JSP Tag

To use an XSLT JSP tag to transform XML documents, perform the following steps:

1. Open the `xmlx.zip` file in the *`BEA Home`*`/wlserver6.0/ext` directory, extract the `xmlx-tags.jar` file, and put it in the `/lib` directory of your Web application, where *`BEA Home`* is the top-level directory in which you installed the WebLogic Server distribution.

2. Add a `<taglib>` entry to the `web.xml` file. For example:

```
<taglib>
  <taglib-uri>xmlx.tld</taglib-uri>
  <taglib-location>/WEB-INF/lib/xmlx-tags.jar</taglib-location>
</taglib>
```

3. To use the tags, add the following line to your JSP page:

```
<%@ taglib uri="xmlx.tld" prefix="x"%>
```

4. Configure the processor. The following procedure shows a generic way to configure the processor:

   a. Enter the following code line to create an `xslt.jsp` file:

```
<%@ taglib uri="xmlx.tld" prefix="x"%><x:xslt/>
```

   b. Register the `xslt.jsp` file in your `web.xml` file, as follows:

```
<servlet>
   <servlet-name>myxsltinterceptor</servlet-name>
   <jsp-file>xslt.jsp</jsp-file>
</servlet>
<servlet-mapping>
   <servlet-name>myxsltinterceptor</servlet-name>
   <url-pattern>/xslt/*</url-pattern>
</servlet-mapping>
```

   c. Put your XML/DTD/XSL documents or servlets in your Web application.

   d. Add an `xslt` prefix to the pathname for the XML document (for example, change `docs/fred.xml` to `xslt/docs/fred.xml`) and then access the document. Because of the `<url-pattern>` entry in the `web.xml` file, WebLogic Server automatically runs the XSLT processor on the XML document and sets the default stylesheet in the document.

   e. To define media type, add code to the JSP to determine the media type for the XML document and the content type for the output.

f. Pass the media type into the `xslt` tag and then set the content type of the response object.

**Note:** The other forms of the XSLT JSP tag are used when stylesheets are not specified in the XML document or your XML stylesheet can be generated inline.

## Example of Using the XSLT JSP Tag in a JSP

The following snippet of code from a JSP shows how to use the XSLT JSP tag to transform XML into HTML or WML, depending on the type of client that is requesting the JSP. If the client is a browser, the JSP returns HTML; if the client is a wireless device, the JSP returns WML.

First the JSP uses the `getHeader()` method of the `HttpServletRequest` object to determine the type of client that is requesting the JSP and sets the *`myMedia`* variable to `wml` or `html` appropriately. Using the `x:xslt` and `x:stylesheet` in unison, if the JSP set the *`myMedia`* variable to `html`, then it applies the `html.xsl` stylesheet to the XML document contained in the *`content`* variable. Similarly, if the JSP set the *`myMedia`* variable to `wml`, then it applies the `wml.xsl` stylesheet.

```
<%
   String clientType = request.getHeader("User-Agent");
   // default to WML client
   String myMedia = "wml";

   // if client is an HTML browser

   if (clientType.indexOf("Mozilla") != -1) {
      myMedia = "http"
   }
%>

<x:xslt media="<%=myMedia%>">
  <x:xml><%=content%></x:xml>
  <x:stylesheet media="html" uri="html.xsl"/>
  <x:stylesheet media="wml"  uri="wml.xsl"/>
</x:xslt>
```

# Using Parsers Other Than the Built-In Parser

This section explains how to use parsers other than the built-in parser. Specifically, it discusses these topics:

- Using the WebLogic Parser Generator to Generate Custom Parsers
- Using Parsers Other Than the Built-In Parser

## Using the WebLogic Parser Generator to Generate Custom Parsers

In a highly scalable Internet application, parser performance can be a bottleneck. To address this problem, WebLogic Server provides the WebLogic Parser Generator. Using the WebLogic Parser Generator you can create a parser that is tailored to a particular DTD. This parser will be much more efficient and faster than a general-purpose parser that can handle any document type. Generated parsers use the SAX API and should be used to parse only XML documents that are based on the DTD used to generate the parser.

### Limitations of Generated Parsers

The following limitations apply to generated parsers:

- Generated parsers support SAX 1.0 features, but not SAX 2.0 enhancements.
- Generated parsers are not fully XML 1.0 compliant; they are specific to a DTD and therefore behave slightly differently from general-purpose, validating parsers. For example, generated parsers do not resolve external entities, including those defined in the DTD.
- A generated parser is able to parse only those documents that conform to the DTD used to generate the parser. A generated parser performs limited validation on the parsed document, and therefore is neither a truly validating nor a non-validating parser.

- Generated parsers ignore `DOCTYPE` statements in an XML document. These statements are not useful because a generated parser operates on the assumption that it was generated with the DTD specified in the `DOCTYPE` statement.
- Generated parsers do not:
  - Support DTD entity references other than the following predefined entity references: `&` (for the ampersand sign), `&quot;` (for a double quotation mark), `<` (for the less-than sign), `>` (for the greater-than sign), and `&apos;` (for a single quotation mark or apostrophe).
  - Support processing instructions, an element content specification of `ANY`, or references to elements that are not defined in the DTD.
  - Report all whitespace or ignorable whitespace in character data.
  - Normalize attribute values to specification.

## Performance and Usage Considerations for Generated Parsers

When deciding whether to use a generated parser, keep in mind the following considerations:

- Parsing performance improvements as much as 10 times that of a standard parser have been observed when generated parsers are used. However, actual performance improvements depend on the application. Generally, the smaller the document, the higher the performance benefit. If performance is critical for your application environment, you should conduct benchmarks on your own application with representative document types.
- The use of generated parsers is intended for applications that require very fast parsing of XML documents and that do not require strict adherence to the XML 1.0 specification.
- Generated parsers are not recommended for parsing documents that attach significance to whitespace, such as prose.

## Generating a Custom Parser

To generate a customized parser, perform the following steps:

1. Make sure the `xmlx.jar` file is in *`BEA Home`*`\wlserver6.0\lib`, where *`BEA Home`* is the top-level directory in which you installed the WebLogic Server distribution.
2. Run the parser generator against the DTD for which you want to generate a custom parser. The following is a typical execution command using the `wml12.dtd` on a Microsoft Windows platform:

   ```
   java weblogicx.xml.parserc -d %SERVER_CLASSES% -root wml
       -package weblogicx.xml.parsers wml12.dtd
   ```

   Where:

   - `-d` designates the destination of the class files (for example, *`BEA Home`*`\wlserver6.0\clientclasses`).
   - `-root` specifies the top-level element—if it is not the DTD name—with the `.dtd` extension deleted.
   - `-package` specifies the package in which the generated parser will be placed.
   - `wml12.dtd` is the name of the DTD.
   - The name of the parser class will consist of the Java class-named version of the root element plus Parser. For example, this example command produces a parser named `wmlparser`.
3. If you want to examine the generated parser's code, use the `-keepgenerated` option.
4. Use the Administration Console to register `wmlparser` in the XML Registry. For instructions about registering a parser in the XML Registry, see "Configuring a Custom-Generated Parser" on page 3-10.

Once this procedure is completed, as long as you use JAXP to parse a WML document, WebLogic Server will use `wmlparser` to do SAX parsing.

# Using Parsers Other Than the Built-In Parser

If you use JAXP to parse your XML documents, the WebLogic Server XML Registry (which is configured through the Administration Console) offers the following options:

- Accept the built-in parser as the server-wide parser.

- Configure a parser other than the built-in parser as the server-wide parser.
- Configure a parser other than the built-in parser as the parser only for applications that use a particular DTD to parse XML documents.
- Configure a custom-generated parser as the parser for particular XML applications.

If you elect to use the API of a general-purpose parser instead of JAXP, you must control all parsing options through the parser's API; the WebLogic Server XML Registry does not control parsing options when the API of a general-purpose parser is used. For example, if you elect to use the built-in parser's proprietary API, rather than JAXP, to parse documents, then you must also use the built-in parser's proprietary API to control all parsing options.

For instructions on how to use the XML Registry to configure parsing options, see "XML Registry Configuration Tasks" on page 3-4.

# Setting Up Local Entity Resolution Using the XML Registry

For instructions about setting up local entity resolution, see "Configuring Local Entity Resolution" on page 3-14.

# 3 XML Administration

The following sections describe XML administration with WebLogic Server:

- Introduction to XML Administration
- Using the XML Registry
- XML Registry Configuration Tasks

For additional information about using the XML Registry to configure WebLogic Server for XML applications, see the *Administration Guide* at http://e-docs.bea.com/wls/docs60/adminguide/index.html and the Administration Console Online Help.

## Introduction to XML Administration

You use the XML Registry, which is accessed through the Administration Console, to configure WebLogic Server for XML applications.

By default, WebLogic Server is configured to use the built-in parser to parse XML documents. As long as you use the default configuration, you do not have to perform any configuration tasks for your XML applications. If you want to use a parser or parsers other than the built-in parser, you must use the XML Registry to configure them.

**Note:** To use the XML Registry to configure WebLogic Server for an XML application, you must use the Java API for XML Parsing (JAXP). If you use a parser through the parser's own proprietary API, the XML Registry has no effect on the behavior of your XML application.

# Using the XML Registry

You create, configure, and use the XML Registry through the Administration Console. You can configure as many XML Registries in a domain as there are servers in a domain, but you can only configure one XML Registry or zero for a particular server.

The benefits of using the Administration Console XML Registry are as follows:

- Configuration changes take effect automatically at run time, provided you use JAXP API.
- When you make changes using the XML Registry, it is not necessary to change the XML application code.
- Entity resolution is done locally. You can use the XML Registry to define a local copy of an entity. This means that the parser does not have to go to the Web site to parse. Instead, the parser downloads the entity from the Administration Server and then caches it.

You can use the XML Registry to specify the following:

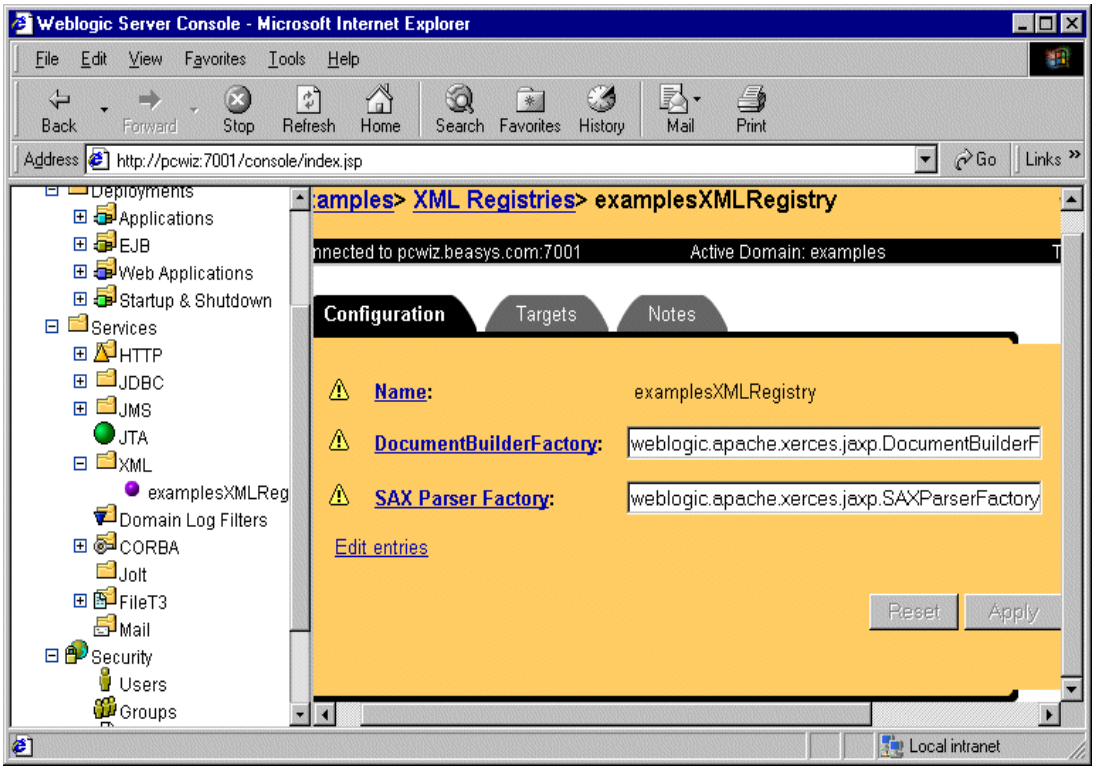
- Specify an alternative default XML parser to use instead of the built-in parser included in this version of WebLogic Server.
- Specify an XML parser to be used to process a particular document type (including custom-generated parsers).
- Specify external entities that are to be resolved using local copies. Once these entities are specified, the Administration Server stores local copies of them in the file system and automatically distributes them to the server's parser at parse time. This feature eliminates the need to construct and set SAX EntityResolvers.
- The advantage of using local copies of external entities is that the parser does not have to consult an external Web site, which improves parser performance.

All of the above capabilities are automatically enabled if your application uses the standard Java API for XML Parsing (JAXP), which is included in this version of WebLogic Server. These capabilities are for use on the server side only.

# Example of an XML Registry

Use the Administration Console to access and edit contents of the XML Registry. Figure 3-1 shows an example of an XML Registry configured through the Administration Console.

**Figure 3-1 Sample XML Registry Code Displayed in Administration Console**




# Relationship Between JAXP and the XML Registry

The XML registry is automatically consulted whenever you use the JAXP API to write your XML applications. The WebLogic Server JAXP implementation can inspect the content of the XML document and delegate to an appropriate parser based on the XML

Registry configuration. If no parser is registered for the document type, the JAXP implementation delegates to the default parsers specified in the XML Registry. If no default parsers are registered, the built-in parser is registered for the document type. Additionally, when WebLogic Server starts, a SAX entity resolver is automatically set so that it can resolve entities that are declared in the registry. As a result, users are not required to modify their XML application code to control the parsers used, or to set the location of local copies of external entities. All of this is controlled by the XML Registry.

**Note:** If you elect to use an API provided by a parser instead of the JAXP API, the XML Registry has no effect on the processing of XML documents.

# XML Registry Configuration Tasks

You can use the XML Registry to perform the following tasks:

- Configuring a Parser Other Than the Built-In Parser

  In this configuration, WebLogic Server uses parsers other than the built-in parser. For example, you can configure WebLogic Server to use the bundled version of the Apache Xerces parser, the Sun parser, and other available parsers.

- Configuring a Custom-Generated Parser

  In this configuration, WebLogic Server uses custom parsers generated using the WebLogic Parser Generator.

- Configuring Local Entity Resolution

  In this configuration, WebLogic Server uses local copies of external entities for entity resolution.

## Using the Built-In Parser

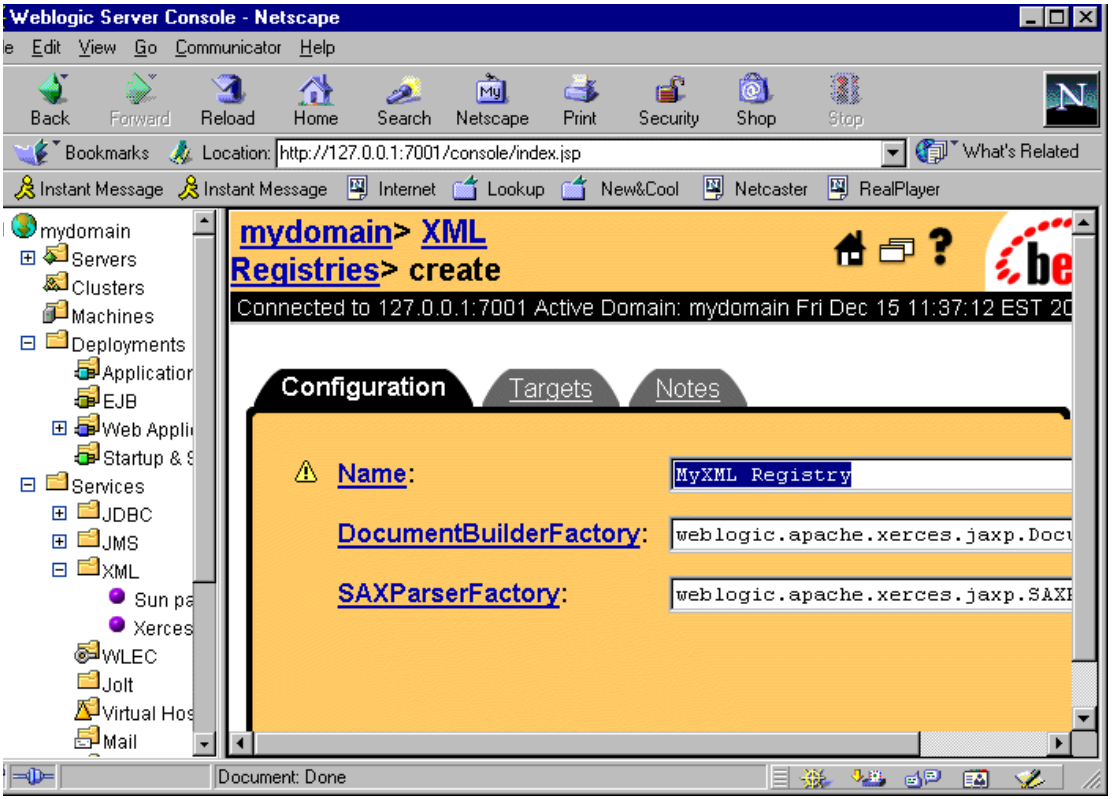
WebLogic Server uses the built-in parser to parse XML documents by default. You do not have to perform any configuration tasks. You simply write and build your XML application using the JAXP API.

# Configuring a Parser Other Than the Built-In Parser

To use a parser other than the built-in parser, use the following procedure:

1. Start the server, open the Administration Console.
2. Right-click the XML node and select Create a new XML Registry from the drop-down menu. The **XML Registries>Create** window is displayed (see Figure 3-2).

**Figure 3-2 XML Registries Create Window**




3. Enter a unique registry name in the `Name` field and set the DocumentBuilderFactory and the SaxParserFactory fields to the appropriate parser classes.

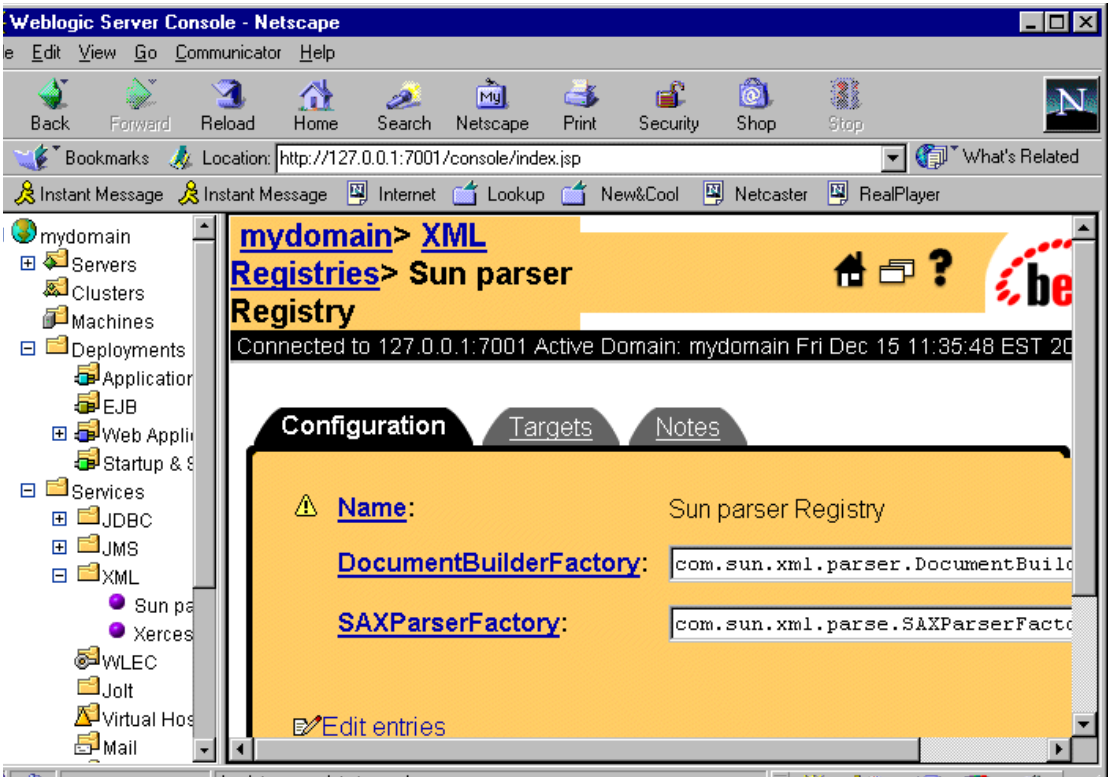
For example, to use the Sun parser, enter the following in the **XML Registries>Create** window (see Figure 3-3):

```
Name: Sun Parser Registry
DocumentBuilderFactory: com.sun.xml.parser.DocumentBuilderFactoryImpl
SAXParserFactory: com.sun.xml.parser.SAXParserFactoryImpl
```

To use the bundled Apache Xerces parser from the XML Module, enter the following in the **XML Registries>Create** window.

```
Name: Xerces parer Registry
DocumentBuilderFactory: "org.apache.xerces.jaxp.DocumentBuilderFactoryImpl"
SAXParserFactory:  "org.apache.xerces.jaxp.SAXParserFactoryImpl"
```

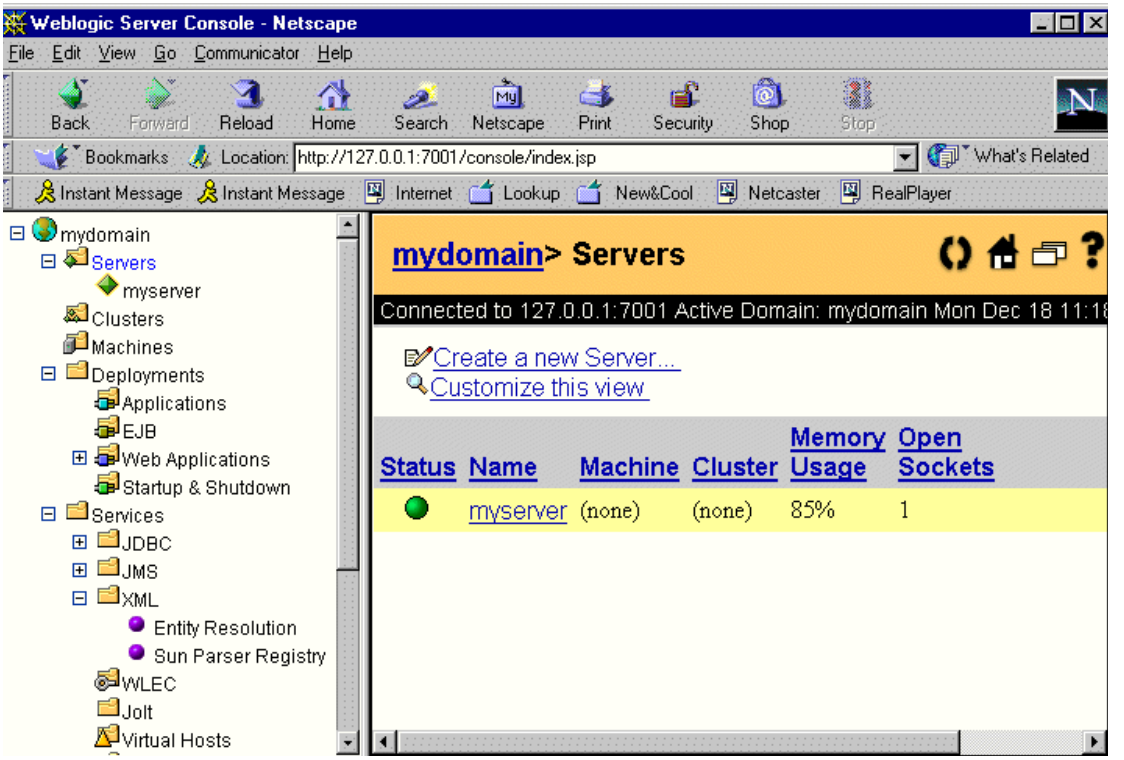
**Figure 3-3 Configured Sun Parser Registry**




4. Click the Create button. The XML Registry is created and listed under the XML node.

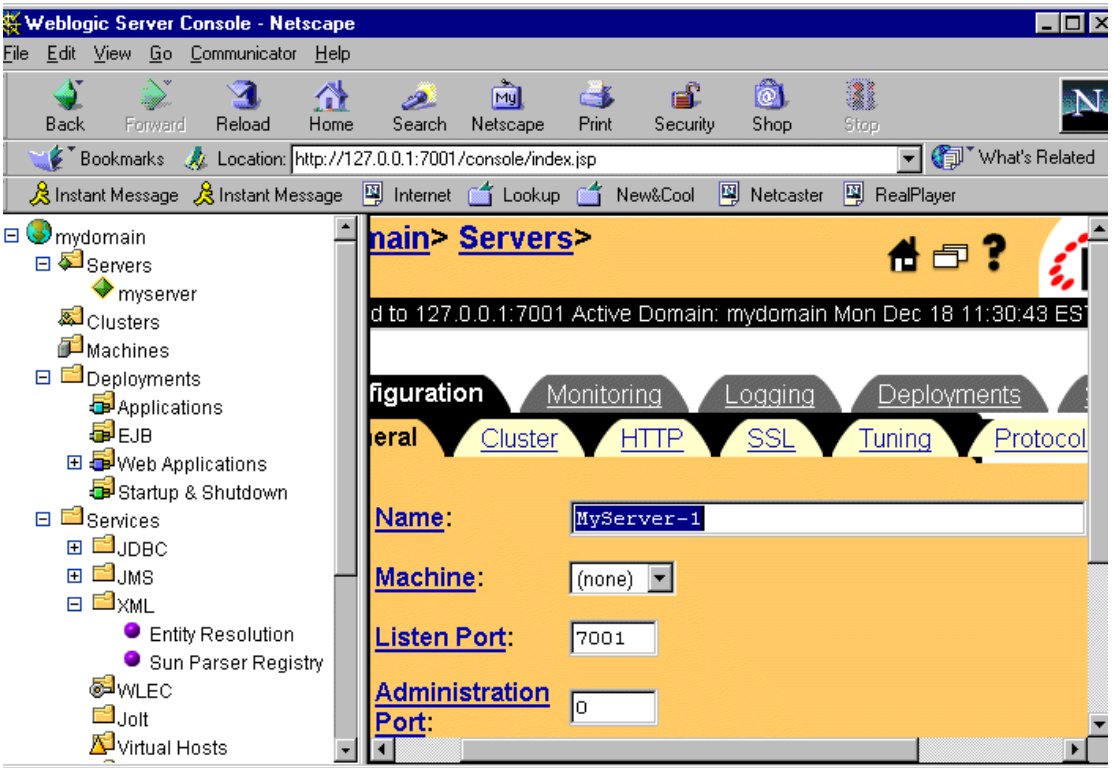
5. Click the Servers node in the left pane. The Servers table displays in the right pane showing all the servers defined in the domain (see Figure 3-4).

**Figure 3-4 Servers Table Window**



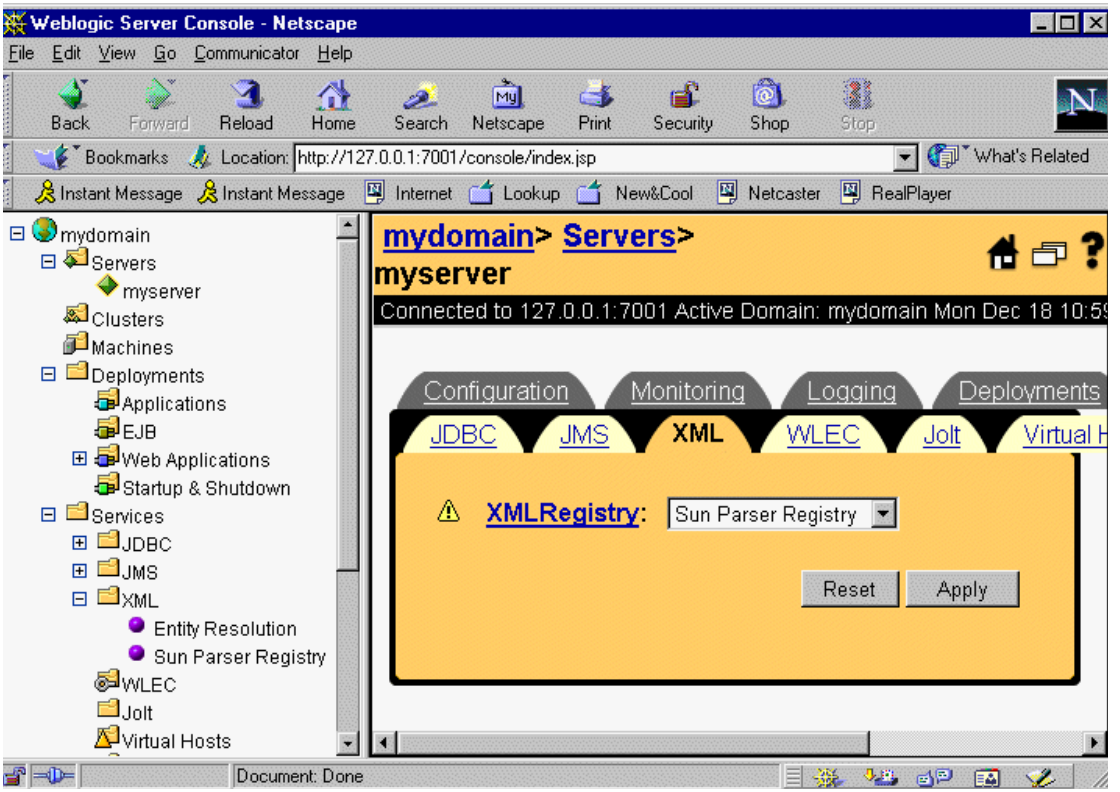

6. Click the Create a New Server text link. A dialog is displayed in the right pane showing the tabs associated with configuring a new server (see Figure 3-5).

**Figure 3-5 Server Configuration Window**




7. Enter values in the Name, Machine, and System Password attribute fields.
8. Click the Create button in the lower right corner to create a server instance with the name you specified in the Name field. The new instance is added under the Servers node in the left pane.
9. Click the Services tab. The XML Registry folder window is displayed (see Figure 3-6).

**Figure 3-6 XML Registry Folder Window**




10. Select the Sun Parser Registry in the XML Registry field and click the **Apply** button.

11. Restart your server so the new settings to take effect.

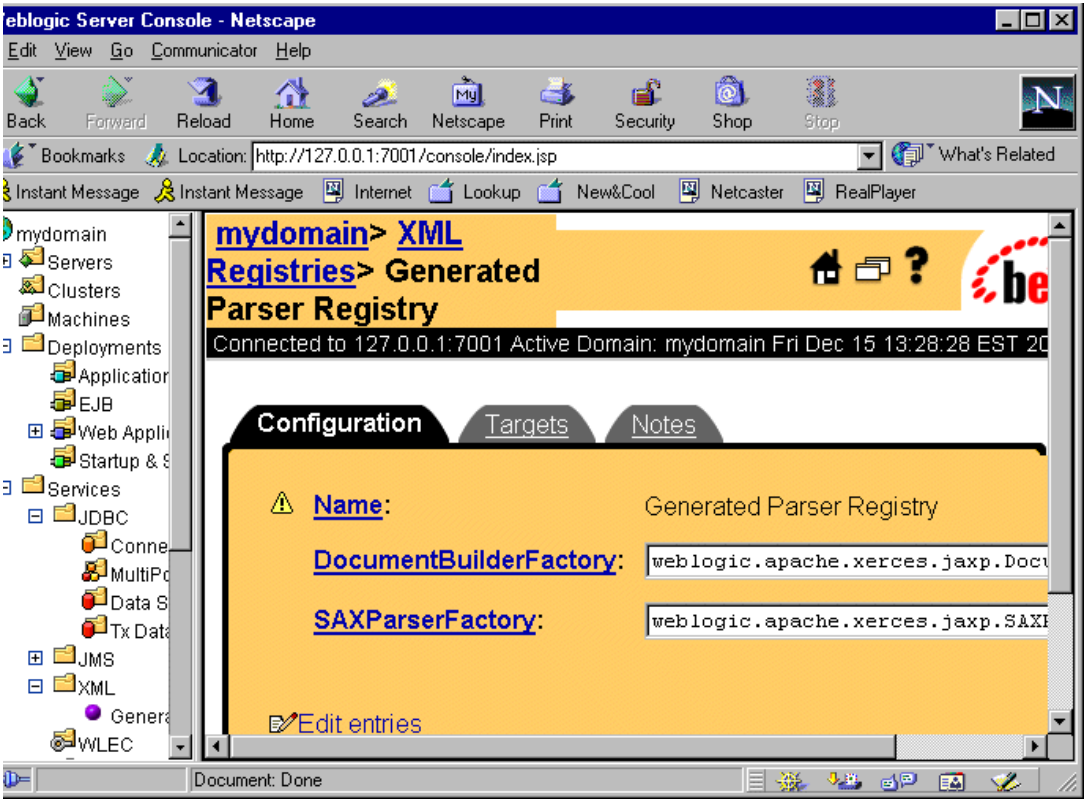
# Configuring a Custom-Generated Parser

You can generate a custom parser based on a specific DTD. For instructions about how to generate a custom parser, see “Using the WebLogic Parser Generator to Generate Custom Parsers” on page 2-16.

To use a generated parser to parse XML documents based on a particular DTD, perform the following steps:

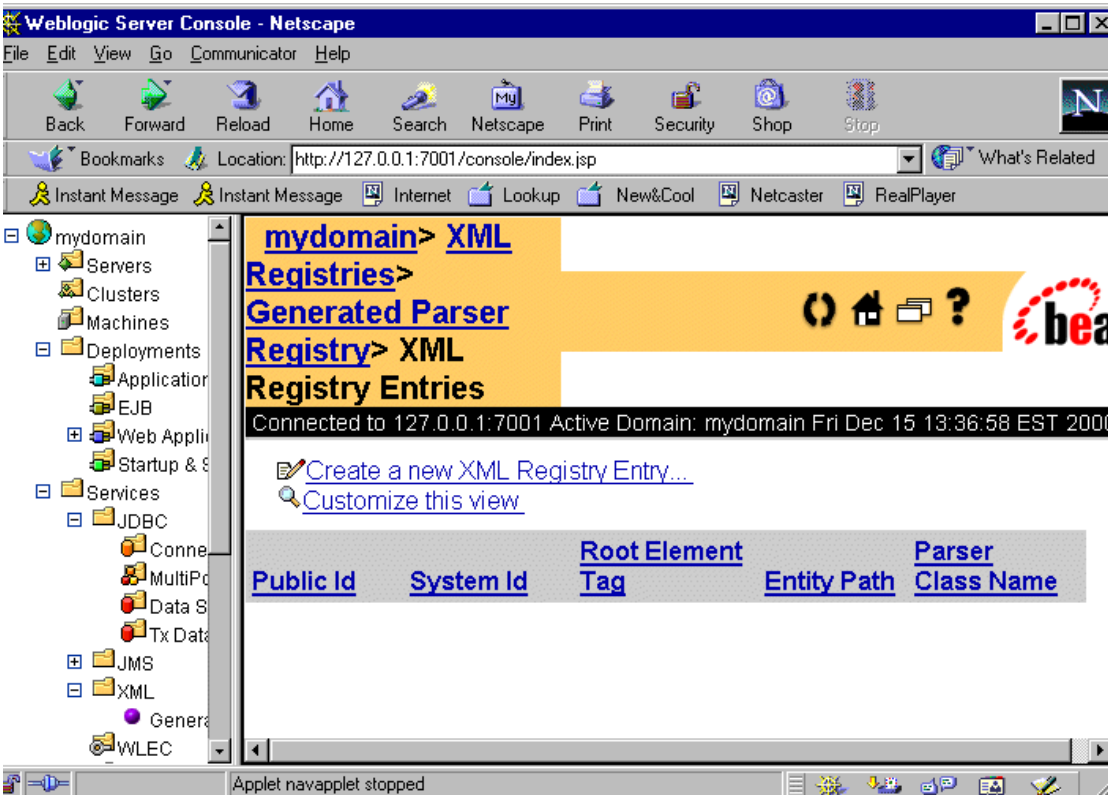
1. Start the server and open the Administration Console.
2. Right-click the XML node and select Create a New XML Registry from the drop-down menu. The **XML Registries>Create** window is displayed (see Figure 3-2).
3. Enter a unique name (for example: Generated Parser Registry) for the new XML Registry and click the Create button. The **XML Registries>Generated Parser Registry** window is displayed (see Figure 3-7).

**Figure 3-7 XML Registries Generated Parser Registry Window**



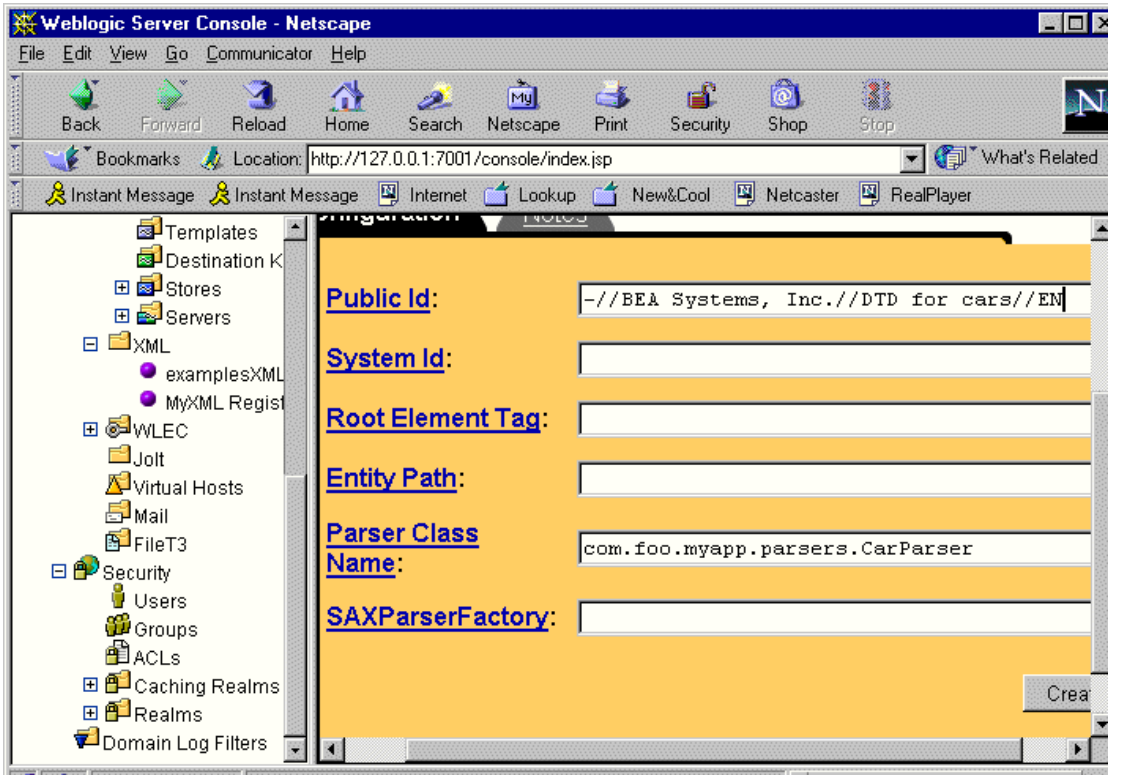

4. Click Edit Entries. The **XML Registry Entries** window is displayed (see Figure 3-8).

**Figure 3-8 XML Registry Entries Window**




5. Click Create a new XML Registry Entry. A blank **XML Registries>Create** window is displayed (see Figure 3-9).

**Figure 3-9 XML Registries Create Window**




6. Enter the doctype in one of the following ways:

   a. Use either the Public Id or the System Id field to specify the doctype. For example, for the car.dtd (see Listing 3-1), enter `-//BEA Systems, Inc.//DTD for cars//EN` in the `Public Id` field.

   b. Specify the Root Element Tag name of the document.

**Listing 3-1 car.xml File**

```
<?xml version="1.0"?>
<!-- This XML document describes a car -->
<!DOCTYPE CAR PUBLIC "-//BEA Systems, Inc.//DTD for cars//EN"
"http://dev/null">
<CAR>
```

```
<MAKE>Toyota</MAKE>
<MODEL>Corrolla</MODEL>
<YEAR>1998</YEAR>
<ENGINE>1.5L</ENGINE>
<HP>149</HP>
</CAR>
```

7. Enter the parser name in the Parser Class Name field. For example, for the `CarParser` generated parser, enter `com.foo.myapp.parsers.CarParser`.
8. Click the Create button. The XML Registry is created and listed under the XML node.

   **Note:** Using URLs is not desirable because both reliability and performance may suffer.
9. Click the Servers node in the left pane to display the Servers table in the right pane, showing all the servers defined in the domain (see Figure 3-4).
10. Click the Create a New Server text link. A dialog is displayed in the right pane showing the tabs associated with configuring a new server (see Figure 3-5).
11. Enter values in the Name, Machine, and System Password attribute fields.
12. Click the Create button in the lower right corner to create a server instance with the name you specified in the Name field. The new instance is added under the Servers node in the left pane.
13. Click the Services tab. The XML Registry folder window is displayed (see Figure 3-6).
14. Select the Generated Parser Registry in the XML Registry field and click the Apply button.
15. Restart your server to make the new settings take effect.

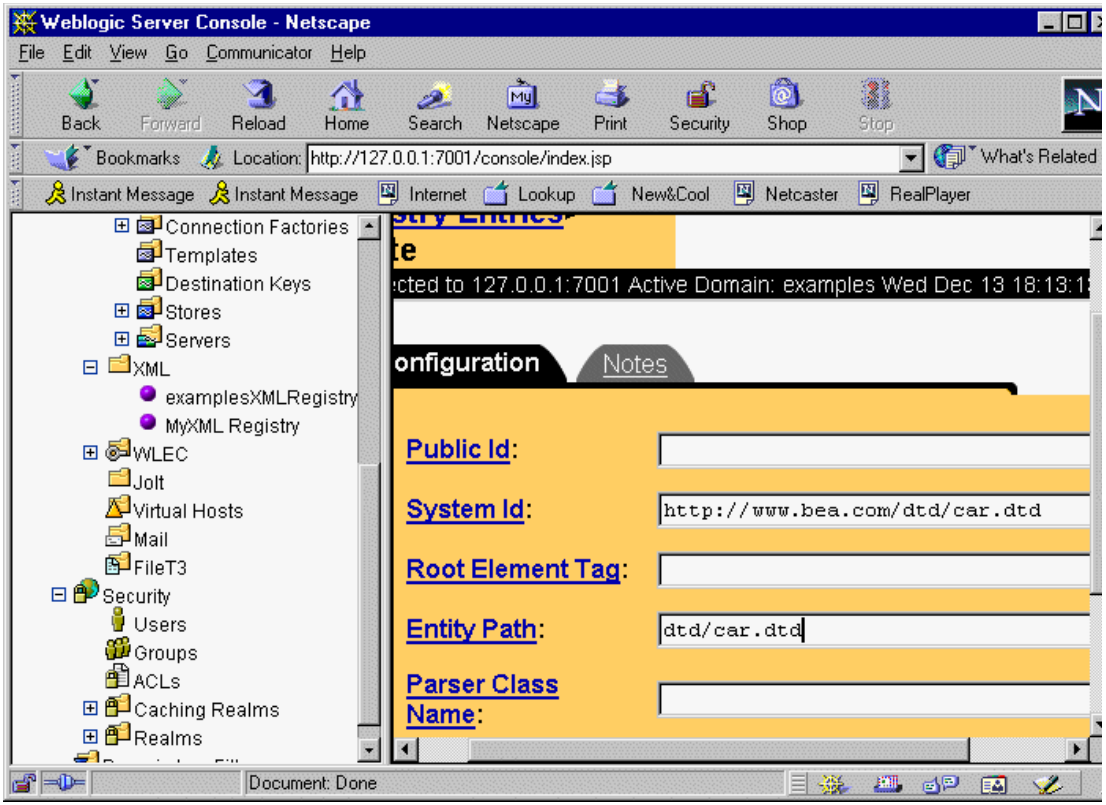
# Configuring Local Entity Resolution

You can use the XML Registry to define local copies of external entities. These entities can then be copied to a local area for resolution so that WebLogic Server does not have to access a remote Web site to resolve them.

To configure local entity resolution, perform the following steps:

1. Start the server, open the Administration Console, and click the XML node. The **XML Registries** window is displayed (see Figure 3-8).
2. Click Create a new XML Registry. The **XML Registries**>**Create** window is displayed (see Figure 3-2).
3. Enter a name for the XML Registry and click the Create button. The named window is displayed (see Figure 3-7).
4. Click Edit Entries. The **XML Registry> Entries** window is displayed (see Figure 3-8).
5. Click Create a new XML Registry Entry. A blank **XML Registry Entries>Create** window is displayed (see Figure 3-10).

**Figure 3-10 XML Registry Entries Create Window for Entity Resolution**




6. Enter the `system Id`. For example, for the `car.dtd` entity (see Listing 3-2), enter `http://www.bea.com/dtd/car.dtd`.

7. In the `Entity Path` field, enter the pathname of the local copy of the entity file in the Administration Server. This pathname must be relative to the registries entity directory. The entity directory is the directory `c:/`*`BEA Home`*`/wlserver6.0/config/examples/xml/registries/`*`reg_name`* in the domain configuration directory, where *`BEA Home`* is the top-level directory in which the WebLogic Server software is installed. The *`reg_name`* is the name of the new XML Registry. For example, for the `car.dtd`, enter `dtd/car.dtd` in the Entity Path field.

8. Click the Create button. The XML Registry is created and listed under the XML node.

9. Copy the entity file into the entity directory. For example, you would copy the `car.dtd` file to `xml/registry/myregistry/dtd/car.dtd`.
10. Restart your server so the new settings to take effect.

# 4 XML Reference

The following sections describe the XML specifications, application programming interfaces (APIs), and tools supported by WebLogic Server:

- Extensible Markup Language (XML) 1.0 Specification
- Simple API for XML (SAX) 1.0
- Document Object Model (DOM) Level 1 API
- W3C XML Namespaces 1.0 Recommendation
- Java API for XML Parsing (JAXP) 1.0.1
- Apache Xerces Java Parser API 1.2.0
- Apache Xalan XML Stylesheet Language Transformer (XSLT) API 1.2
- Additional Resources

## Extensible Markup Language (XML) 1.0 Specification

The W3C Recommendation for XML provides the following abstract:

**"**The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML."

The Recommendation is available at the following URL:

http://www.w3.org/TR/REC-XML

The complete XML specification is available at http://www.w3.org/TR/REC-xml.html.

# Simple API for XML (SAX) 1.0

The SAX API is platform-independent and language-neutral. It is a standard interface for event-based XML parsing that was developed collaboratively by the members of the XML-DEV mailing list.

SAX applications process an XML document by creating a parser object and associating handlers with XML events. Once these tasks are done, the parser can read through the document as events occur, and pass them to the handlers. Events represent the entities within the document, such as start of document, end of document, start of element, and end of element. The SAX interface provides a simple coding model and is useful for processing XML documents with a relatively simple hierarchical structure. Hence, the SAX interface provides what is required by the bundled parser to parse XML documents. For links to the Javadoc for the SAX API, see "Additional Resources."

# Document Object Model (DOM) Level 1 API

The DOM API is a platform- and language-neutral interface It allows programs and scripts to access and update the content, structure, and style of XML documents dynamically. DOM gives you access to the information stored in your XML document as a hierarchical object model, much like a tree with the document's root element as the tree's root node. Using the DOM interface, you can access different parts of XML documents, navigate through them, and make changes and additions to them.

When an application invokes a DOM parser, the parser processes the entire document, creating an in-memory object model, which the application can process in any fashion it chooses. The DOM approach is most useful for more complex documents because it does not require a developer to interpret every element. For links to the specification and Javadoc for DOM, see "Additional Resources" on page 4-5.

# W3C XML Namespaces 1.0 Recommendation

WebLogic Server does support namespaces; however, in this release of WebLogic Server, the XML Registry does not support namespaces.

The following abstract is taken from the W3C XML Namespace Recommendation:

**"**XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by Universal Resource Identifier (URI) references."

The XML Namespaces 1.0 Recommendation is available on the Internet at `http://www.w3.org/TR/1999/REC-xml-names-19990114/`.

# Java API for XML Parsing (JAXP) 1.0.1

JAXP is Sun's Java API for XML parsing. JAXP provides basic support for parsing and manipulating XML documents through a standardized set of Java platform APIs. Thus, applications that use JAXP to process XML documents are portable across platforms.

**Note:** The JAXP API does not replace either the SAX or DOM API. Instead, it adds some convenience methods that are designed to make the SAX and DOM APIs easier to use.

JAXP 1.0.1 consists of the `javax.xml.parsers` package. This package contains two vendor-neutral factory classes: `SAXParserFactory` and `DocumentBuilderFactory`. The JAXP 1.0.1 specification describes these classes as follows:

- "The SAXParserFactory defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents."
- "The DocumentBuilderFactory defines a factory API that enables applications to configure and obtain a parser to parse XML documents into a DOM document tree."

The JAXP specification is available on the Internet at `http://java.sun.com/xml/`. The JAXP Javadoc is available at `http://java.sun.com/xml/docs/api/`.

# Apache Xerces Java Parser API 1.2.0

The Apache Xerces Java Parser 1.2.0 package includes the API documentation for SAX and DOM, the two most common interfaces for programming XML. In addition, the parser provides documentation for classes that are not part of the SAX and DOM API's, but are useful for writing parser programs. For links to the specification and Javadoc for the Xerces parser, see "Additional Resources" on page 4-5.

# Apache Xalan XML Stylesheet Language Transformer (XSLT) API 1.2

The Apache Xalan-Java version 1.2 XSLT processor is used for transforming XML documents. It implements the W3C Recommendation 16 November 1999 XSL Transformations (XSLT) Version 1.0. XSLT is a stylesheet language for transforming XML documents into other XML documents, HTML documents, or other document types. The language includes the XSL Transformation vocabulary and XPath, a language for addressing parts of an XML document. An XSL stylesheet describes how to transform the tree of nodes in the XML input into another tree of nodes.

For links to the specification and Javadoc for the Xalan XSLT processor, see "Additional Resources" on page 4-5.

**Note:** To use the XSLT processor, you must supply your own language extensions.

# Additional Resources

This section lists various resources that are available online to help you learn about programming with WebLogic XML:

- Code Examples
- Related WebLogic Services
- General XML Information
- Tutorials and Online Courses
- XML APIs
- XML Specifications

## Code Examples

XML code examples and supporting documentation are included in the WebLogic Server distribution at *BEA Home*`\wlserver6.0\samples\examples\xml`, where *BEA Home* is the directory in which the WebLogic Server software is installed. The default installation directory is *c:*`\bea\wlserver6.0\samples\examples\xml`.

## Related WebLogic Services

- Programming WebLogic Enterprise JavaBeans at *http://e-docs.bea.com/wls/docs60/ejb/index.html*.
- Programming WebLogic JMS at *http://e-docs.bea.com/wls/docs60/jms/index.html*.
- Programming WebLogic JSP at *http://e-docs.bea.com/wls/docs60/jsp/index.html*.
- Programming WebLogic HTTP Servlets at *http://e-docs.bea.com/wls/docs60/servlet/index.html*.

- Using WAP with WebLogic Server at `http://e-docs.bea.com/wls/docs60/wap/index.html`.

# General XML Information

- W3C (World Wide Web Consortium) at http://www.w3c.org.
- XML.com at http://www.xml.com.
- XML FAQ at http://www.ucc.ie/xml/.
- XML.org, The XML Industry Portal at http://www.xml.org/.
- W3C: Extensible Stylesheet Language at http://www.w3.org/Style/XSL/.
- XLS Frequently Asked Questions at http://www.laidback.org/dpawson/xsl/xslfaq.html.

# Tutorials and Online Courses

- A Technical Introduction to XML at http://www.xml.com/pub/a/98/10/guide0.html.
- XML Authoring Tutorial at http://www.xml.com/pub/r/32.
- Working with XML and Java at http://java.sun.com/xml/tutorial_intro.html.
- Tutorials for using the Java 2 platform and XML technology at http://developerlife.com/.
- Developing XML Solutions with JavaServer Pages Technology at http://java.sun.com/products/jsp/html/JSPXML.html.
- XML, Java, and the Future of the Web at http://www.xml.com/pub/a/w3j/s3.bosak.html.
- Chapter 14 of the XML Bible: XSL Transformations at http://metalab.unc.edu/xml/books/bible/updates/14.html.
- XSL Tutorial by Miloslav Nic at http://zvon.vscht.cz/HTMLonly/XSLTutorial/Books/Book1/index.html.

- XML Schema Part 0: Primer at http://www.w3.org/TR/2000/CR-xmlschema-0-20001024/.

## XML APIs

- Apache Xerces Java Parser at http://e-docs.bea.com/wls/docs60/xerces/index.html.
- Apache Xalan XSLT Transformer at http://e-docs.bea.com/wls/docs60/Xalan/index.html.
- SAX (Simple API for XML) at http://e-docs.bea.com/wls/docs60/xerces/index.html.
- DOM (Document Object Model) at http://e-docs.bea.com/wls/docs60/xerces/index.html.
- JAXP (Java API for XML Parsing) at http://java.sun.com/xml/docs/api/.

## XML Specifications

- Extensible Markup Language (XML) 1.0 Specification at http://www.w3.org/TR/REC-xml.html.
- Extensible Stylesheet Language (XSL) 1.0 Specification at http://www.w3.org/TR/xsl/.
- DOM (Document Object Model) Level 2 Specification at http://www.w3.org/TR/DOM-Level-2/.
- Java API for XML Parsing (JAXP) 1.0.1 Specification at http://java.sun.com/xml/download.html.
- JSR-000031 XML Data Binding Specification at http://java.sun.com/aboutJava/communityprocess/jsr/jsr_031_xmld.htm.
- XML Path Language (XPath) Version 1.0 Specification at http://www.w3.org/TR/xpath.
- XML Linking Language (XLink) Specification at http://www.w3.org/TR/xlink.

- XML Pointer Language (XPointer) Specification at http://www.w3.org/TR/WD-xptr.
- Namespaces in XML Specification at http://www.w3.org/TR/REC-xml-names/.
- XML Schema Part 1: Structures at http://www.w3.org/TR/xmlschema-1/.
- XML Schema Part 2: Datatypes at http://www.w3.org/TR/xmlschema-2/.

# Index

## T



## U



## W



## X