

Tutorial: Recursive XSLT

This tutorial describes sample code that uses recursive XSLT calls and JavaScript to display an expanding and collapsing tree view of an XML document.

Contents

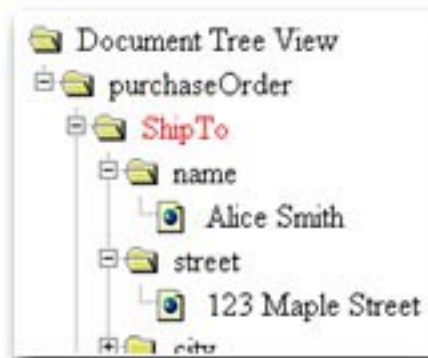
1. [Concepts](#)
2. [Design](#)
3. [Required Software](#)
4. [Setup](#)
5. [Implementation](#)
6. [Resources](#)
7. [Feedback](#)





Concepts

One of the strengths of XML as a language for data exchange is its flexibility. An XML document can be transformed into many text formats, including HTML, PDF, and WML. This tutorial uses [XML](#), [XSL](#) and [JavaScript](#) to create an interactive view of a purchase order. The combined code adds icons and user interface widgets that a user can click to expand or collapse branches of a tree-style view (also called outline view) of the data in an XML document.



About XML

XML stands for **eXtensible Markup Language**. Like HTML, XML is a subset of SGML (Structured Generalized Markup Language), optimized for delivery over the Web. Unlike HTML, which tags elements in Web pages for presentation by a browser, e.g. `<bold>Oracle</bold>`, XML tags elements as data, e.g. `<company>Oracle</company>`. For example, you can use XML to give context to words and values in Web pages, identifying them as data instead of simple textual or numeric elements.

About XSL

Unlike HTML, XML can keep the instructions for presenting data separate from the data itself. The XML tags define the structure and content, and then a stylesheet is applied to it to define the presentation. XML data can be presented in a variety of ways (both in

appearance and organization) simply by applying different stylesheets to it. For example, a different interface can be presented to different users based on user profile, browser type, or other criteria by defining a different stylesheet for each different presentation style. Or, stylesheets can be used to transform XML data into a format tailored to the specific application or device that receives and processes the data. Stylesheets may be applied either on the server or on the client.

About JavaScript

Netscape's [documentation](#) describes JavaScript this way:

"JavaScript is Netscape's cross-platform, object-oriented scripting language. JavaScript is a small, lightweight language; it is not useful as a standalone language, but is designed for easy embedding in other products and applications, such as web browsers. Inside a host environment, JavaScript can be connected to the objects of its environment to provide programmatic control over them."





Design

The sample code for this tutorial was designed to demonstrate the following parsing techniques:

- Using recursive calls in an XSL stylesheet.
- Invoking JavaScript code from an XSL stylesheet.





Required Software

The following software is required to build and run this tutorial.

- Java 2 Platform, (Standard Edition or Enterprise Edition) that includes Java Runtime Environment (JRE) 1.4. Available for download from <http://java.sun.com/downloads.html>. The Enterprise Edition (J2EE) is also included with [Oracle9i JDeveloper](#), which OTN members can [download for free](#).
- XDK Java Components in the Oracle [XDK for Java](#), available for download from OTN and included with Oracle9i JDeveloper.
- Tutorial source code and supporting files ([treeview.jar](#), 12.3 KB)

See the [Setup](#) section for information about installing and running the tutorial.





Setup

This section explains the steps to install and configure the tutorial. It assumes that you have installed and configured the software described in the [Required Software](#) section.

1. Download [treeview.jar](#), a compressed Java Archive containing source code and other files.
2. Open a command window and change directories to make the download directory active. For example, if you downloaded `treeview.jar` to `d:\tutorials`, change directories to `d:\tutorials`.
3. Make sure that the executable file `jar.exe` (included with the Java 2 Platform) is in your environment's path. For example, the following command adds the path to a `jar.exe` that resides in `d:\jdevFolder\jdk\bin`.

```
set path=%path%;d:\jdevFolder\jdk\bin
```

4. From the command line, enter the following to unpack the archive. This command creates a directory named `ex20030215_treeview` that contains subdirectories and source files.

```
jar -xvf treeview.jar
```

5. Change directories to make the `ex20030215_treeview\src` directory active.

```
cd ex20030215_treeview\src
```

6. Set your `CLASSPATH` to include the path to your Java compiler (`javac.exe`) and the path to the Oracle XML Parser library (`xmlparserv2.jar`). For example,

```
set CLASSPATH=.;..;d:\jdevFolder\jdk\bin;d:\jdevFolder\lib\xmlparserv2.jar
```

7. Enter the following command to invoke the XDK tool that takes an XML document (`pol_ann.xml`) and applies an XSL stylesheet (`treeview.xsl`) to generate an HTML file (`result.html`).

```
java oracle.xml.parser.v2.oraxsl pol_ann.xml treeview.xsl result.html
```

8. Now you can load the generated HTML file (`result.html`) in a Web browser to display the results of the processing.
-





Implementation

This section lists the following key parts of the sample application:

- [XML Input](#)
- [XSL Code](#)
- [JavaScript Code](#)
- [HTML Output](#)

XML Input

The following XML code from `po1_ann.xml` defines the data for a purchase order, including shipping and billing addresses and a list of items ordered.

```
<purchaseOrder orderDate="1999-10-20">
  <ShipTo isPicked="1" country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
  </ShipTo>
  <BillTo isPicked="1" country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </BillTo>
  <LineItems isPicked="1">
    <LineItem partNum="872-AA">
      <ProductName>Lawnmower</ProductName>
      <Quantity>1</Quantity>
      <USPrice>148.95</USPrice>
    </LineItem>
  </LineItems>
</purchaseOrder>
```

```

    <comment>Confirm this is electric</comment>
</LineItem>
<LineItem partNum="926-AA">
    <ProductName>Baby Monitor</ProductName>
    <Quantity>1</Quantity>
    <USPrice>39.98</USPrice>
    <ShipDate>1999-05-21</ShipDate>
</LineItem>
</LineItems>
<comment>Hurry, my lawn is going wild!</comment>
</purchaseOrder>

```

XSL Code

The XSL code below comes from `treeview.xsl`. It produces an HTML file that includes calls to JavaScript code implemented in `treemenu.js`. Notice, too, that the code defines a template named `treeview` that includes the element `<xsl:call-template name="treeview">`. Thus this code uses recursion to process the input data and create a hierarchical tree representation. The code also uses `<xsl:when test="number($depth)=0">` to record recursive depth and `<xsl:when test="$content/*">` to control the processing flow of the recursion.

...

```

<xsl:template name="treeview">
    <xsl:param name="depth"/>
    <xsl:param name="content"/>
    <xsl:choose>
        <xsl:when test="number($depth)=0">
            foldersTree = gFld("Document Tree View", "")
            <xsl:for-each select="$content/*">
                <xsl:call-template name="treeview">
                    <xsl:with-param name="content" select="."/>
                    <xsl:with-param name="depth" select="number($depth)+1"/>
                </xsl:call-template>
            </xsl:for-each>
        </xsl:when>
        <xsl:otherwise>
            <xsl:choose>
                <xsl:when test="$content/*">
                    <xsl:choose>

```

...

JavaScript Code

This sample application uses JavaScript to enable an otherwise static page to respond to UI events, making the page interactive. For example, the following code from `treemenu.js` defines responses when a user clicks on a folder icon (representing a category of information such as Ship To data) or a document icon (representing a data item such as a City name).

```
function clickOnFolder(folderId)
{
    var clicked = indexOfEntries[folderId]
    if (!clicked.isOpen)
        clickOnNode(folderId)
    return
    if (clicked.isSelected)
        return
}

function clickOnNode(folderId)
{
    var clickedFolder = 0
    var state = 0
    clickedFolder = indexOfEntries[folderId]
    state = clickedFolder.isOpen
    clickedFolder.setState(!state) //open<->close
}
```

HTML Output

The file [result.html](#) shows the HTML code generated when the sample application processes the data in `po1_ann.xml`. The generated code includes calls to functions implemented in the JavaScript file `treemenu.js`.





Resources

This tutorial is part of a series, *Oracle XML Parser Techniques*. Following are links to resources that can help you understand and apply the concepts and techniques presented in this tutorial. See the [Required Software](#) section to obtain the tutorial source code and related files.

Resource	URL
OTN XML Center	http://otn.oracle.com/tech/xml/content.html
Oracle XDK for Java	http://otn.oracle.com/tech/xml/xdk_java/
Oracle XML DB	http://otn.oracle.com/tech/xml/xmldb/content.html
XML in Oracle Database Applications	http://otn.oracle.com/tech/xml/htdocs/xml_in_oracle_apps.htm
Customizing Data Presentation	http://otn.oracle.com/tech/xml/htdocs/xml_custom_presentation.htm





Feedback

If you have questions or comments about this tutorial, you can:

- Post a message in the [OTN Sample Code discussion forum](#). OTN developers and other experts monitor the forum.
- Send email to the author. <mailto:Robert.Hall@oracle.com>

If you have feedback about the XDK, please send email to:

- <mailto:Jinyu.Wang@oracle.com>

If you have suggestions or ideas for future tutorials, please send email to:

- <mailto:Raghavan.Sarathy@oracle.com>

