Extending the Reach of Your Applications
A simple example using IBM WebSphere Everyplace Suite

Tim Robinson, Ph.D.
Senior Software Engineer, IBM
January 2001

Tim Robinson describes a very simple application that was developed in the IBM Solution Partnership Center lab to test and illustrate concepts that are central to WebSphere Everyplace Suite. WebSphere Transcoding Publisher, a component of WebSphere Everyplace Suite, is used to customize content (using XSLT stylesheets) for multiple device types. Tim includes a detailed discussion about the stylesheet used for generating the wireless markup language (WML).

WebSphere Everyplace Suite is an environment for building services and applications for pervasive computing devices. Because it complies with Internet standards and open-application architecture, integrating it into your applications is very straightforward.

This article describes a sample application designed to talk to WAP-based phones, personal computers, and PDAs with Web browsers. In many pervasive computing application scenarios, you need to keep one set of back-end business logic and use content adaption to tailor the user interface correctly for each client.

The application described here is a temperature converter that prompts the user for a temperature and the units (Celsius or Fahrenheit) and then converts the temperature into the other units. This is a very simple example that combines several technologies: JavaServer Pages (JSPs) and Java code for content creation and program logic, XML for data formatting, and XSL for content adaption. I assume you have a general understanding of Web-based applications, markup languages, and XML specifically.

I'll start with an overview of the application, then provide all the sources for the application and basic installation instructions. You can download the application code at http://www-106.ibm.com/developerworks/library/ibm-extend/wesdemo.zip .

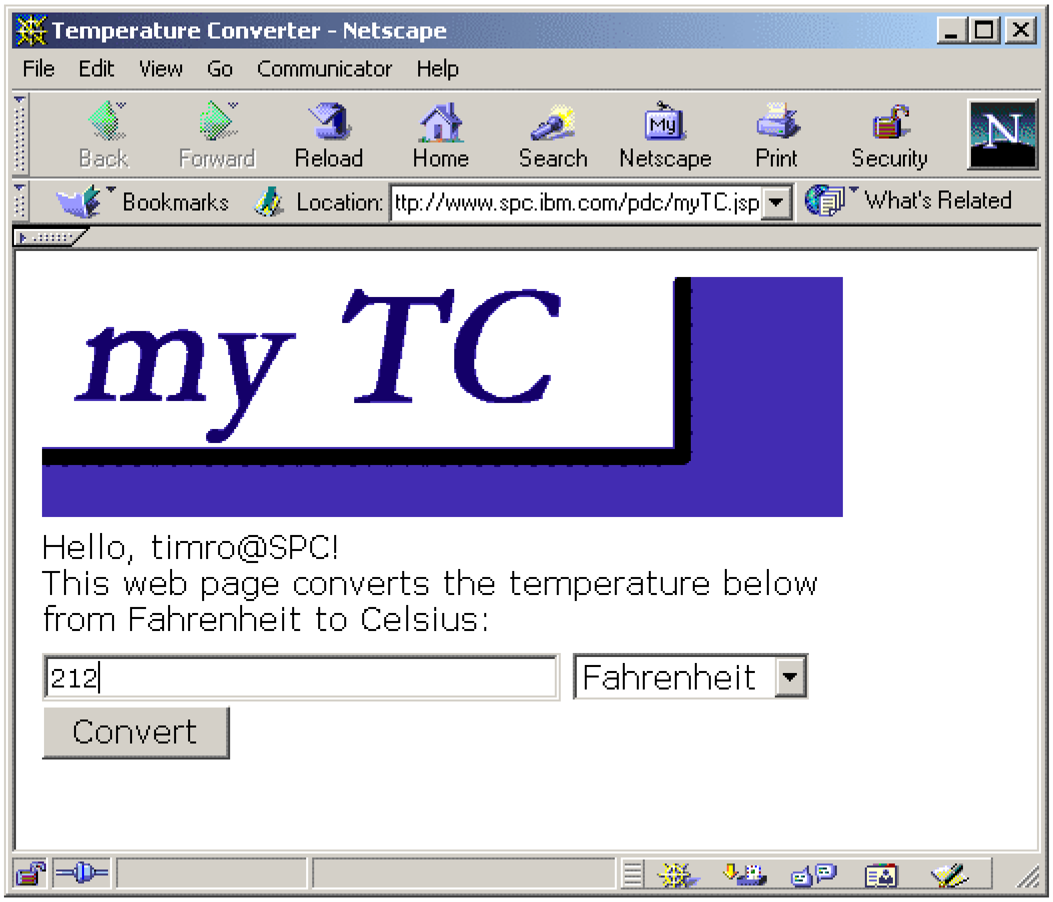
Using WebSphere Transcoding Publisher

This application relies heavily on WebSphere Transcoding Publisher, which is a component of WebSphere Everyplace Suite. Transcoding Publisher converts standard HTML content to markup formats that are compatible with a spectrum of pervasive device types. Here, it's used to customize the query front-end of the application. Transcoding Publisher also supports advanced content conversion, using XSLT style sheets, to render XML data for display on target devices. This application uses content conversion to build the response of the converted temperature.

The application in action

Before going any further, take a look at the application in action. Here's the screen that is displayed when the user of a regular Web browser connects to the application:

Figure 1.


Temperature Converter - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Bookmarks Location: ttp://www.spc.ibm.com/pdc/myTC.jsp What's Related
my TC

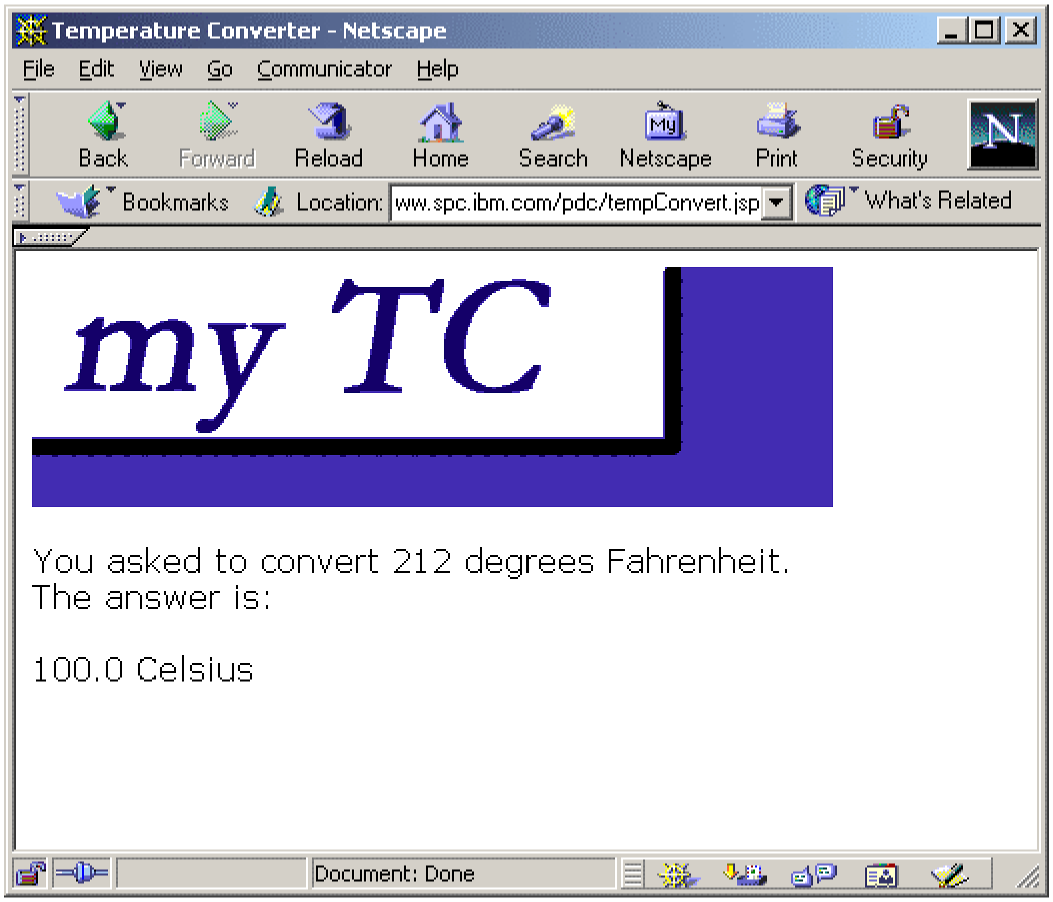
Hello, timro@SPC!
This web page converts the temperature below from Fahrenheit to Celsius:

212 Fahrenheit

Convert

After entering a temperature and selecting the units, the user clicks Convert and the following screen appears:

Figure 2.


Temperature Converter - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Bookmarks Location: ww.spc.ibm.com/pdc/tempConvert.jsp What's Related
my TC
Document: Done

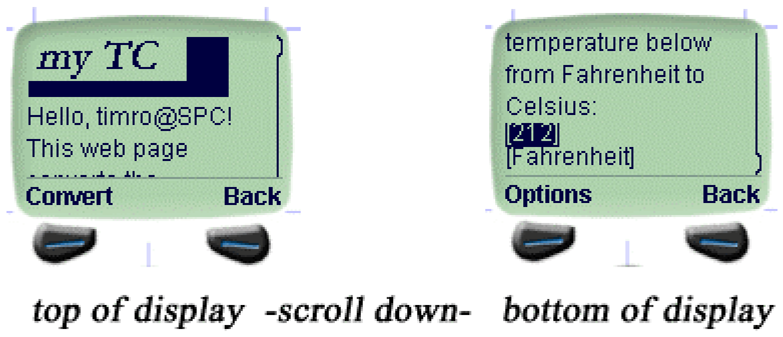
You asked to convert 212 degrees Fahrenheit.
The answer is:

100.0 Celsius

The application on a mobile phone

Here's how the application looks on a WAP phone simulator. Notice how Transcoding Publisher has re-sized the image and converted it to a format that can be displayed by the phone. Transcoding Publisher has also altered the markup so that all the text and the data entry elements are present in the phone browser. However, the user now has to scroll down the screen to get to where the temperature input is displayed. (You can trim the content that's displayed with the Text Clipping API, but that topic is beyond the scope here; see the sidebar).

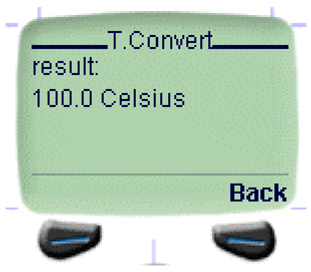
Figure 3.


my TC
Hello, timro@SPC!
This web page

temperature below
from Fahrenheit to
Celsius:
[212]
[Fahrenheit]

Again, the user enters a temperature and gets back a very simple response, which is tailored exactly to the screen capabilities through an XSL style sheet:

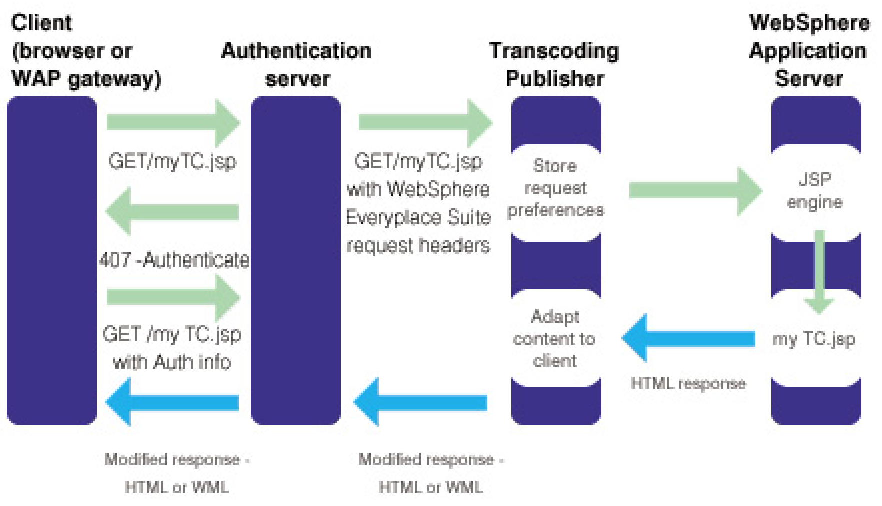
Figure 4.


T.Convert
result:
100.0 Celsius

Web Request flow for the data entry page
Now that I've gone over the basics of the application, I'll dissect the actual components. I'll start with a flow diagram of the application's first step: handling the request for the initial page. I'll include relevant code snippets in this article. If you want all the source, you can download it from http://www-106.ibm.com/developerworks/library/ibm-extend/wesdemo.zip . You can use your favorite zip-file handler or jar to extract the components (they are not packaged in a subfolder). Instructions on where to place the component files are provided in the readme.txt file in the package.

When the browser (HTML or WML) requests the file myTC.jsp, here is what is happening to the Web request inside WebSphere Everyplace Suite:

Figure 5.


Client (browser or WAP gateway)
Authentication server
Transcoding Publisher
WebSphere Application Server
GET/myTC.jsp
407 -Authenticate
GET /my TC.jsp with Auth info
GET/myTC.jsp with WebSphere Everyplace Suite request headers
Store request preferences
JSP engine
my TC.jsp
Adapt content to client
HTML response
Modified response - HTML or WML
Modified response - HTML or WML

The request travels through a network (wired or wireless), reaches the Everyplace Wireless Gateway (if the request is from a WAP phone or wireless client device), and then proceeds to the Everyplace Authentication Server. If the mobile device is using a third-party gateway (or if the request is from a traditional PC-based Web browser), the request instead travels through the Internet directly to the Authentication Server.

At this point, the request from the browser agent is examined for WebSphere Everyplace Suite credentials. If these are missing, the Everyplace Wireless Gateway or the Authentication Server generates an HTTP Authenticate request, which causes the browser to prompt the user for a WebSphere Everyplace Suite domain user name, realm, and password. (The user name and realm are combined in the response with an @ sign. This is the basis for the typical user identifier inside WebSphere Everyplace Suite).

Once authenticated, the Authentication Server is ready to pass on the Web request to the downstream application servers; but first, it modifies the HTTP header of the request with information that can be used by the other application servers that are in the same security region as the WebSphere Everyplace Suite domain. You can find more information on these concepts in the Authentication Server and security sections of the IBM Redbook An Introduction to IBM WebSphere Everyplace Suite: Accessing Web and Enterprise Applications (see Resources).

In Figure 5, the next application server hop for the Web request is a server running Transcoding Publisher in Web-proxy mode. For test and demonstration purposes, you can simply proxy all traffic from the Everyplace Authentication server to Transcoding Publisher. In deployment environments, it is important to send only traffic requests that require content adaption through Transcoding Publisher. Once Transcoding Publisher receives the request, it processes it through a set of input filters, then places a direct query to the application server for the Web content (in this case, the JSP myTC.jsp).

Identifying the WebSphere Everyplace user

Upon receipt of the request, the JSP is run in WebSphere Application Server. This JSP generates HTML. Embedded in the JSP is an example of how to use a JSP scriptlet to extract user identity information from the HTTP header information that the Authentication Server adds to each HTTP request. More sophisticated applications could use this information in a number of ways. The Tivoli Personalized Services Manager component provides a Java and JavaBean toolkit for building portals based upon users' preference selections. Or, a JavaBean could record the user identity for accounting in pay-by-the-click applications. This example isn't that sophisticated. Instead, the user and WebSphere Everyplace Suite realm information is put into the output of the Web page through an expression evaluation. Here's the code:

Java scriptlet in JSP page

```
 <% String pvcuser = request.getHeader("x-ibm-pvc-user");
if (pvcuser == null) pvcuser="non-WES user"; %>
Hello, <%=pvcuser%>!
```

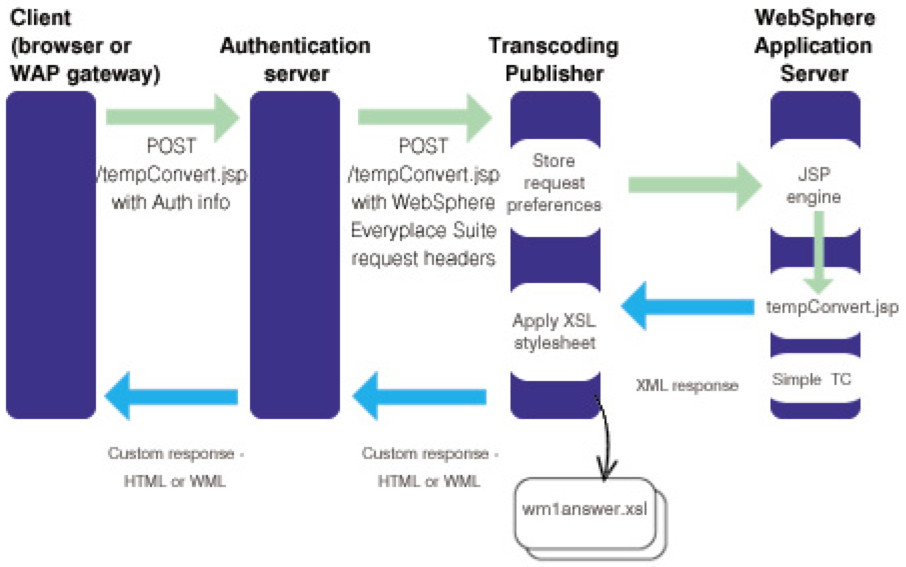
If you are running in a non-WebSphere Everyplace Suite environment, the HTTP header x-ibm-pvc-user might not be set, so in that case, I set the pvcuser string to something obvious, rather than leaving it empty. The rest of this JSP is pretty straightforward.

The application server returns the HTML generated by the JSP to Transcoding Publisher. At this point, Transcoding Publisher processes the HTML page, based upon the device that has requested the content. For full-function PC browsers, it doesn't significantly modify the content. For WML browsers, it converts the HTML into WML, re-sizes the graphic image, and reduces the bitdepth. I was careful to choose a graphic that retains the important structural logo components upon transformation. After WebSphere Transcoding Publisher is done, the request flows back through the Authentication server and Wireless gateway to return to the requesting device.

<u>Application flow - getting the answer</u>

Now I'll look at the second half of the application, converting the input value, and returning that to the user:

Figure 6.


Client (browser or WAP gateway)
Authentication server
Transcoding Publisher
WebSphere Application Server
POST /tempConvert.jsp with Auth info
POST /tempConvert.jsp with WebSphere Everyplace Suite request headers
Store request preferences
JSP engine
tempConvert.jsp
Simple TC
Apply XSL stylesheet
XML response
Custom response - HTML or WML
Custom response - HTML or WML
wm1answer.xsl

Because the end-user device was already authenticated by WebSphere Everyplace Suite upon submitting the reply to the form, there is no additional sign-on screen. This time, the Web request flows through the Authentication Server and Transcoding Publisher, and to the application server, where the tempConvert.jsp is run. Unlike the previous JSP, this JSP generates output in XML. The JSP is simple and short enough to include here in its entirety.

tempConvert.jsp

```
   <?xml version="1.0" ?>
   <!DOCTYPE answer SYSTEM "answer.dtd" >
   <%@ page language = "java"%>
   <%  // set up response type in a mini scriptlet
        response.setContentType("text/xml");
   %>
   <jsp:useBean id="tempconv" scope="page"
class="com.ibm.spc.SimpleTC"/>
   <jsp:setProperty name="tempconv" property="*" />
   <!-- here's the meat of the XML document-->
   <answer>
      <title>Temperature Converter</title>
      <lmessage>You asked to
      convert <jsp:getProperty name="tempconv" property="temp" />
      degrees <jsp:getProperty name="tempconv" property="units" />.
      The answer is:</lmessage>
      <smessage>result:</smessage>
      <result><jsp:getProperty name="tempconv"
property="result"/></result>
   </answer>
```

When JSPs need to incorporate nontrivial processing logic, they can embed JavaBeans. The JSP used here (tempConvert.jsp) does this for the temperature conversion. The JSP creates an instance of the JavaBean, sets properties of the JavaBean using the input from the form, and then places the result inside an XML element. You can see the source for the JavaBean in the file com/ibm/spc/SimpleTC.java. This JavaBean was developed using VisualAge for Java, and it can be viewed in that or any text editor. The XML that is generated by tempConvert.jsp is formatted according to a custom document type definition (DTD), answer.dtd. This very simple DTD suits the needs of this application. In addition to the result element -- that is, the converted temperature -- the DTD incorporates two elements that are selected by the XSLT style sheet depending on device type. For devices with ample display space there is a <lmessage> tag around the long message. Another tag, <smessage>, is for brief content that is better suited for the small screen dimensions of a phone.

One of the XSLT style sheets

WebSphere Transcoding Publisher uses an XSLT style sheet to convert the XML output from the JSP into either an HTML or WML document. For devices that WebSphere Transcoding Publisher identifies as capable of displaying WML content, this style sheet is applied:

Source of wmlanswer.xsl

```
   <?xml version="1.0"?>
   <wml xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xsl:version="1.0">

     <!-- ... comments from source snipped -->

     <template>
     <!-- this makes sure that our card(s) have a "back" button -->
       <do type="prev"><prev/></do>
     </template>

     <card id="card" title="T. Convert">
       <p>
         <xsl:value-of select="answer/smessage"/>
         <br/>
         <xsl:value-of select="answer/result"/>
       </p>
     </card>
   </wml>
```

This style sheet uses a "simplified" style sheet format (see sidebar). In XSLT Programmer's Reference, Michael Kay describes four design patterns for XSLT stylesheets (see Resources). This style sheet implements the "fill-in-the-blanks" pattern. In this pattern, the XSLT language is used to extract the desired information from the source XML document (in this case, the output from tempConvert.jsp) into a standard document template. Although there are other cases where the full power of the XSLT language is needed, this application simply needs to pull the correct information from the JSP response.

To create this style sheet, I used a standard editor to author the WML page. When designing the WML page, comments are added as placeholders for content that will be inserted from the source XML document. After work is completed with the WML editor, the basis for the response page is built. This consists of the top level <wml> element, the <template> element, the <card>, and some elements familiar to those who have used HTML.

Inserting XSLT instructions into a page of markup

The source style sheet above has all of the XSL-related markup additions in red. The template WML document has comments for the text message from the application and the value of the conversion. Next, <xsl:value-of select="answer/smessage"> and the <xsl:value-of select="answer/result"> instructions are inserted. The first of these selects the short message from the XML response and the other selects the value from the result element. This example uses very simple XPath syntax in the select attribute for each of the instructions. Inside the top level WML element, the xmlns:xsl="http://www.w3.org/1999/XSL/Transform" and xsl:version="1.0" attributes are added so that the template can be used as a simplified style sheet by an XSLT processor. Finally, the <!DOCTYPE ... > tag that was added by the WML editor between the XML document declaration and the top level XML element to define the DTD is removed because I've augmented the original WML document format by adding the XSL namespace attributes to the <wml> tag. The XSLT style sheet to generate the HTML version of the page is similar, except that it contains markup that includes the logo image and the cascading style sheet, and it selects the long message from the back-end program.

On your own

You can investigate further by installing this sample application in your environment. You will need a WebSphere Everyplace Suite environment that is up and running, including the addition of some users to the Tivoli Personalized Services Manager (TPSM) environment. Because this Web application is very simple, for testing purposes the JavaServer Pages and JavaBean can be installed on the same WebSphere Application Server that is used for TPSM, under the default WebSphere Application. All files except the JavaBean source and class file are in the top level (no folders or subpaths) of the zip file. Begin by copying the .jsp and .gif files to HTTP Server document root and the SimpleTC.class file into: usr/WebSphere/AppServer/hosts/default_host/default_app/servlets/com/ibm/spc (you'll need to create this directory). It is also a reasonable idea to fully qualify the post URL in the file myTC.jsp to the correct fully qualified URL for your environment. In some deployment scenarios, this isn't necessary, but in the interest of space, I won't delve into those details.

The HTML content that is delivered by the JSP files references a cascading style sheet (called Master.css). This style sheet needs to be placed in a subdirectory of the HTTP Server document root called "theme." On the WebSphere Transcoding Publisher system, you should copy the .xsl files into the IBMTrans directory and the .dtd file into the IBMTrans/etc subdirectory (because the XML file references the DTD by way of the SYSTEM path). After copying these style sheets, you need to register them through the WebSphere Transcoding Publisher Administration Console.

It is also possible to run this application in an environment that only has a JSP-enabled Web application server and WebSphere Transcoding Publisher. In that configuration, the user identification in myTC.jsp will quietly fail, because the x-ibm-pvc-user HTTP header will not be set.

Resources

XSLT Programmer's Reference, by Michael Kay (Wrox Press, 2000), provides a very useful discussion and reference of the XSLT language for the programmer.

Learn more about WebSphere Everyplace Suite [link to http://www-3.ibm.com/pvc/products/wes/index.shtml] and WebSphere Transcoding Publisher [link to http://www-3.ibm.com/pvc/products/wes/solutions5.shtml] elsewhere on this site.

Browse the WebSphere Transcoding Publisher library [link to http://www-4.ibm.com/software/webservers/transcoding/library.html] for a collection of information and whitepapers about WebSphere Transcoding Publisher.

Learn how to create and test wireless applications with the WebSphere Everyplace Suite SDK [link to http://www-3.ibm.com/pvc/tech/wes_sdk.shtml].

Refer to the developerWorks XML zone [link to http://www-106.ibm.com/developerworks/xml/] for information about using XML.

For the latest on the Wireless Application Protocol, visit the WAP Forum. [link to http://www.wapforum.org]

For information on XML, see W3C Architecture domain. [link to http://www.w3.org/XML]

For information on XSL, see W3C User Interface domain.[link to http://www.w3.org/Style/XSL]

About the author

Tim Robinson has spent several years working in computing engineering fields covering topics in scientific/technical computing, Internet applications architecture, and network infrastructure and security. In the IBM Solution Partnership Center in San Mateo, he has helped Independent Software Vendors (ISV) port and test their applications on IBM RS/6000 AIX platforms, IBM's Java technology, and IBM's Internet-based application and e-commerce servers. Recently, Tim has been working as a Pervasive Computing Technical Consultant involved in running a world-wide testing community (based in the IBM Solution Partnership Centers) for application developers.

Text Clipping

The Transcoding Publisher Text Clipping API lets you (or a Java programmer) extract the content you want from the HTTP response. These clippers are used on HTML content that has been converted to WML; you can use the clippers to display just part of the page on the WML device. Text Clippers can work with either a text representation of the HTTP response or with a Document Object Model (DOM) representation. Transcoding Publisher includes sample Text Clippers for both styles.

Simplified Style Sheets

In the XSLT specification, there is a provision for something called **Literal Result Element As Stylesheet**. This mouthful is commonly known as a simple or simplified style sheet. If you've authored XSLT style sheets, the sample here probably looks like it's missing some elements. The simplified style sheet construct allows for markup in more familar template languages to be augmented with XSLT elements. This is accomplished by including the XML namespace reference for XSLT elements in the outermost element. By using the simplified style sheet approach, you can create most of the graphical and user interface layout using customary authoring tools, and then insert the necessary XSLT elements to extract the appropriate data from the source XML document.