AS/400 Business Data Goes Mobile Thanks to XML

by LindaMay Patterson & George Weaver

PartnerWorld for Developers, AS/400

June 2000

Table of Contents

Introduction 1
The Basics 1
PvC (Pervasive Computing Device) 1
eXtensible Markup Language (XML) Basics 4
The Application Scenario 5
The Application Architecture 6
Application Implementation 9
courses_xml 9
CourseServlet 12
CourseXML 15
CourseSelection Stylesheets 16
Small Device Course Selection 23
AbstractServlet 23
AbstractXML 24
In Conclusion 26
For Additional Information 27
Appendix A: WML and WAP 28
Appendix B: CourseSelection files 29
Trademarks 32
For more information 33
About the Author 33
For more information about PartnerWorld for Developers, AS/4(. 33
Production 33

Introduction

The world of computing is constantly changing both from a hardware and software perspective. Pervasive Computing devices (PvC) are one of the latest types of devices to emerge on the scene. These devices have put the mobile into mobile computing. These devices, ranging from cell phones to WebTV, can be used to conduct business by connecting via the Internet or a private network to business applications running on an AS/400™ server.

eXtensible Markup Language (XML) plays an important role in this new mobile environment. XML, used to define portable data, provides both the basis for the various XML dialects used by these devices and the tagged business data displayed on these devices. The transformation of the business data (represented in an XML dialect) into a form tuned to the device is accomplished by using eXtensible Stylesheet Language (XSL) and a stylesheet transformer, like IBM's LotusXSL.

Palm Device

The AS/400 system supports communications with PvC devices and personal computing desktop browsers by using various enablers. The AS/400 system is being extended even further into the Internet environment. The WebSphere Application Server on AS/400 provides an entry point into a business application. XML and the XML related enablers (available on the AS/400 system) provide the data and formatting capabilities to interact with these devices. The objective of this paper is to show how a variety of devices can access data stored on an AS/400 system. This paper shows a sample application, its architecture, and drills down into the code necessary to build and run the application. The application consists of Java Server Pages (JSP) and servlets running in the WebSphere application server, Java, XML, and various XML enablers.

The Basics

PvC (Pervasive Computing Device)

Pervasive computing can be defined as enabling any device (for example — PC, cell phone, information appliance) to interact with any network (i.e. private network, Internet, wired or wireless) and to access data or applications in either a consumer or business oriented fashion. Pervasive computing devices (including hand held computing devices) are those devices used in the manner described above. Hand held computing devices have greatly evolved over the years — from standalone limited function address book and memo pad (storage and retrieval) devices to network connected, personal computer equivalents capable of accessing business data and applications. These advances have been fueled by the continued innovation in a number of key areas such as screen and battery technologies, miniaturization of processor and memory resources, and communications — wired and wireless.

Devices in the PvC category include IBM Work pad, Palm Pilot type devices, and other non Palm Operating System based devices (such as those based on Windows CE or EPOC). They may have small keyboards or have fairly full featured browsers. Cell phones, another example of a PvC device, are now capable of accessing web oriented applications. This is a second

generation technology in some European countries that have supported Global System for Mobile Communications (GSM) wireless network infrastructure for several years. Today in the US, coverage is generally limited to large metropolitan areas. Most of the effort is under the auspices of the Wireless Application Protocol (WAP) consortium, which has defined a platform, network, and device independent reference model for enabling wireless phone based data services.

So, how do these devices interact with business applications? They may attach by secure LAN, dial up Internet Service Provider, wireless network provider, or cable TV modem to the network. Some application environments may require security; others may not. In the case of wireless communications, the end users will typically contract service through a third party network provider (often the regional telephone company), which must be securely connected to this enterprise Information Technology (IT) infrastructure. Figure 1 depicts this web application environment.

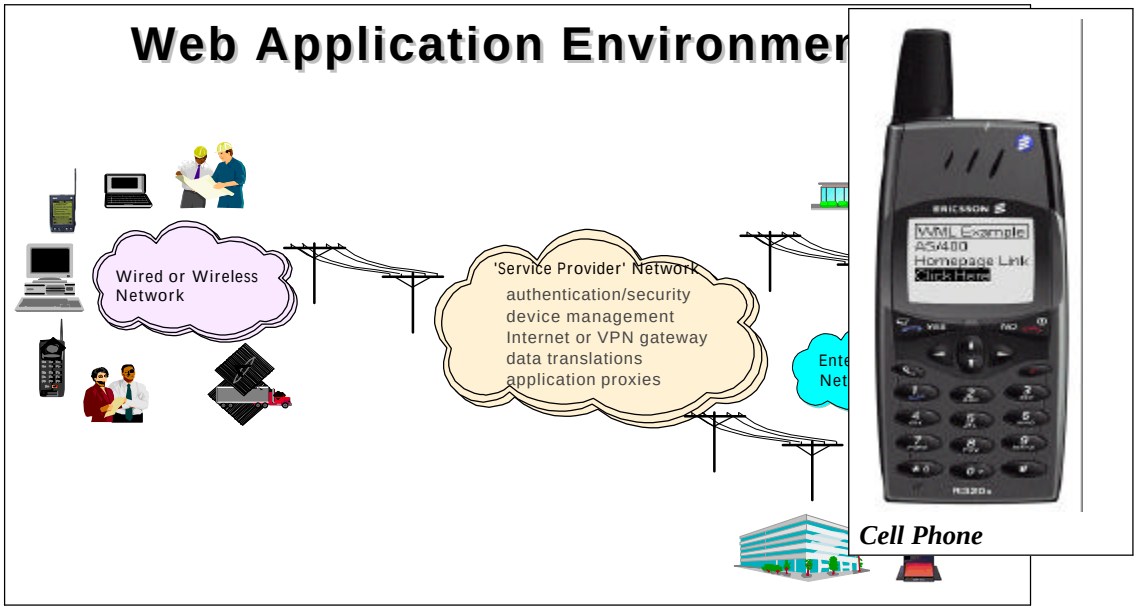


Figure 1 Web Application Environment

There are two key challenges inherent to developing applications to communicate with PvC devices:

1. Each device has a different size presentation space and input mechanism.
2. The languages supported by the device may be different or have unique considerations.

The viewing space dictates the amount of information that can sensibly be presented on these devices. For example, cell phones have under two square inches of viewing space, palm devices have under four square inches of viewing space, while a PC desktop browser can have the full display. The different input mechanisms affect the dialogue structure and limit the way responses are made. Many of these devices support a subset of HTML with their own tags mixed in. Cell phone manufacturers are using Wireless Markup Language (WML) as the standard markup language. WML is a dialect that contains some HTML and some unique tags. A WML document is composed of one or more cards to carry out the dialog with the cell phone user. A card would equate to an application display screen.

Today, numerous popular web sites have 'Palm friendly' web pages tailored for small displays, and some European sites even have WAP/WML specific pages. Typically, these sites deploy device specific URLs to access tailored content. In other words, they generally maintain multiple copies of essentially the same content. Certainly, this gives an optimal solution for the end user but comes at a price. Content providers must build and maintain x versions of the same material for x different device families, which is costly in terms of resources and time to market. A more affordable approach is to represent the data once, as a XML document and use XML enablers to transform the data appropriately and dynamically, based upon the requesting device and/or browser.

Some Wireless Application Protocol (WAP) gateway vendor products can dynamically format existing web page content (HTML and graphics) into WML, which is then compiled into a compressed byte stream that is sent from the WAP gateway to the phone device's micro browser. Similarly, these products enable user input from the WML browser by translating from the compiled WML sent to the gateway to WML, and sending it to the web server resources. Additionally, products such as the IBM WebSphere Transcoding Publisher can not only do this for WML devices, but also can customize HTML and graphics content for devices such as Palm Pilots. These may be acceptable solutions in some instances. However, using a web application server, XML, and stylesheets approach has many distinct advantages:

- Using one of the multi-purpose products mentioned above may consume additional processor resources and require additional hardware and software to be purchased, configured and administered.
- WebSphere, XML and the XML enablers (used in this example) are available on AS/400 at no additional cost.
- WebSphere and the XML enablers run natively on AS/400 and consume relatively modest resources.
- Stylesheets allow more granular control over the formatting options.

For this application, a master XML document and XML enablers were used to transform the data to the form needed by the device. This application shows flexibility by communicating with a variety of PvC devices and a fully functional PC desktop browser. The devices were separated into three categories:

1) PC desktop browser,

2) Palm (hand held) devices including webTV, and

3) Cell phone enabled by Wireless Access Protocol.

Working with these devices, their features and limitations put demands on the application design. The limitations of the devices and the variety of languages used made representing the master business data as XML and using XML enablers to transform the data a necessity.

eXtensible Markup Language (XML) Basics

XML is a specification used to create specific self describing markup languages. HTML and XML are similar in that they are both used to create documents consisting of tagged data (<tag>). At the same time, they are very different because HTML consists of tags focusing on the presentation of data within a browser environment, while XML supplies the ground rules for creating a tag based vocabulary appropriate for the associated data. An XML dialect consists of a set of meaningful tags that define each piece of data for a particular document or domain.

Representing the business data as an XML document, then transforming the data to fit the needs of the requesting device, ensures flexibility and simplicity while maintaining a single version of the master data. XML plays an important role in this new mobile environment. XML supports this new environment in two ways:

1. Used to represent tagged business data that can be transformed to the particular grammar used by the device and
2. By providing the basis specifications for creating new markup languages used by these devices.

Various XML enablers aid this new mobile environment including eXtensible Stylesheet Language (XSL) and the LotusXSL processor. For details on XML and XML enablers, see the reference section at the end of this document.

The Application Scenario

This application allows a client using either a cell phone, palm device, webTV, or PC browser to access the course catalog to get information about the courses that are available. The client device type was used to determine the tagging of the information and the amount of information returned. The dialog for a wireless device is shown in Figure 2.

1. Client selects the Course Catalog Application.
2. Server sends a selection criteria page.
3. Client makes selection for:
    a) Course Category — Java or XML
    b) Day of week — Monday or Tuesday
    c) Difficulty/Level — Introductory, Intermediate or Advanced
4. Server receives course query request and returns a list of matches.
5. Client requests abstract for specific course.
6. Server receives course abstract request and returns the abstract.
7. Client can:
    a) Select a different course,
    b) End dialog, or
    c) Make a different request.

Figure 2: Normal wireless device scenario

The dialog for a PC browser session consists of the steps shown in Figure 3. The dialog is simplified because all pertinent information is returned after the first request, made possible because of the available display space.

1. Client selects the Course Catalog Application.
2. Server sends a selection criteria page.
3. Client makes selection for:
    a) Course Category — Java or XML
    b) Day of week — Monday or Tuesday.
4. Server receives course query request and returns a list of matches.
5. Client can go back to selection criteria.

Figure 3: PC browser scenario

Before reviewing the application, let's look at a course from the courseX.xml file, shown in Figure 4 on the next page. By using the xml file as input, the master course catalog DB2 table does not have to be accessed to create an XML document on the fly. One way to ensure consistent data between the XML document and the DB2 table is to regenerate the .xml file when the database is updated. (This consideration is beyond the scope of this white paper.)

The course catalog contains specific information about each course offered at a multi-track seminar/event. Figure 4 shows the content of a course within the catalog. Each XML document is defined as a specific document type which contains specific Course entries. (This document

is type Courses.) Each course is composed of the course details (course identification, duration, level/difficulty, short title, long title, abstract, and presenter information) and schedule information (day, time and location). The XML document has a short title and a long title to better deal with the size limitations of the various devices.

Figure 4 Example of XML document

```
<Course TopicArea="XML" Type="Lecture">
  <CourseDetails>
   <CourseID>X102</CourseID>
   <Duration>1.0</Duration>
   <Level>Introductory</Level>
   <ShortTitle>Create XML</ShortTitle>
   <LongTitle>Fundamentals of creating XML</LongTitle>
   <Abstract>How to create XML documents using the associated
enablers.</Abstract>
   <Presenter>
       <Name>Ted Brown</Name>
       <Company>IBM</Company>
       <EMail>Ted.Brown@us.ibm.com</EMail>
   </Presenter>
  </CourseDetails>
  <Schedule>
    <Occurance>
     <Day>1</Day>
     <Time Period="AM">10:00</Time>
     <Location>
       <Building>Cedar</Building>
       <Room>D110</Room>
     </Location>
```

The Application Architecture

While defining the application architecture, Java, JSP, servlets, WebSphere application server, and various XML enablers were leveraged. Figure 5 on the next page depicts the application components. The various devices that can access the application are represented in a hexagon. The Java classes are defined in the rectangles. The XSL Stylesheets used to filter and transform the courseX.xml document are shown in the rounded corner rectangles. For this example, the Apache Xerces XML Parser and the IBM LotusXSL processor were used to customize the device content.

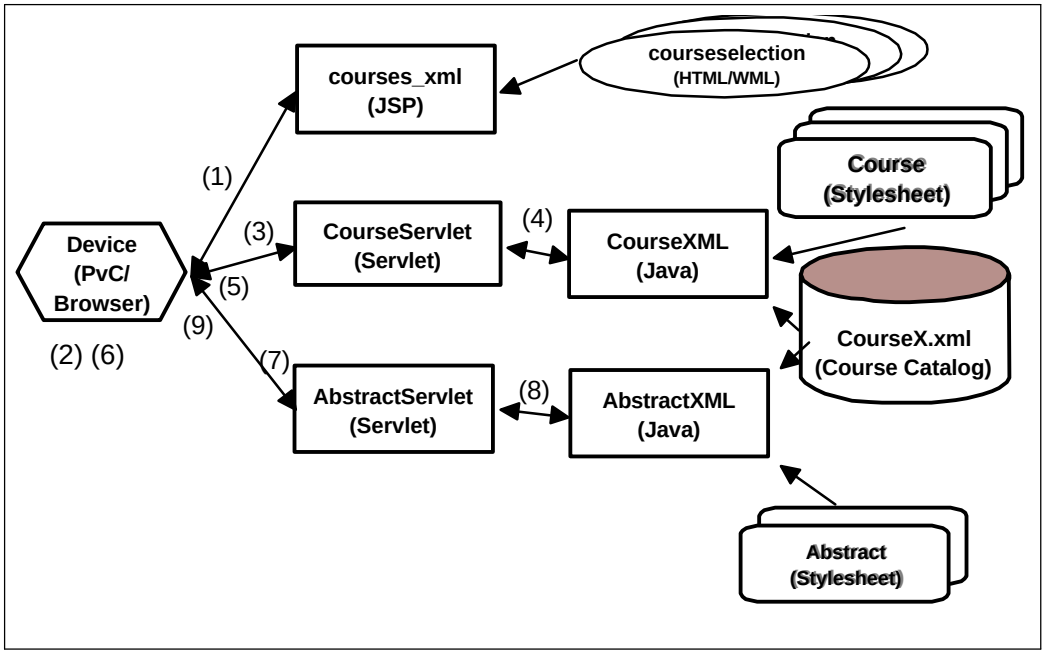


Figure 5 Application Components

The interaction between the various components (depicted in Figure 5) is as follows:

1. The device loads the appropriate “home page” when the user selects the Courses option from the menu (device links to the URL for the application which invokes the Course_xml.jsp).
2. The device user follows the query dialog, determining the course selection criteria.
3. The CourseSelection servlet is invoked, which performs the following:
    a. Determines the device type making the request,
    b. Obtains the selection criteria (course of study, day, and possibly level), and
    c. Creates an instance of the CourseXML class, passing it the selection criteria and the device language (determined from the device type).
4. The CourseXML class does the following:
    a. Determines the stylesheet to use (by the device language),
    b. Parses the selected stylesheet and input XML document creating Document Object Model (DOM) objects,
    c. Dynamically modifies the stylesheet with the selection criteria,
    d. Invokes the LotusXSL processor (input the stylesheet and XML document DOM objects), creating a new DOM object that represents the query results, and
    e. Returns the resulting DOM object to the CourseSelection servlet.
5. The Course Selection servlet transforms the DOM Object into a print stream which is then sent back to the device (** end of dialog if a desktop PC Browser **).

6. The device user views the results and can request a course abstract.
7. The AbstractServlet is invoked, which:
    a. Determines the device making the request,
    b. Obtains the course number selected, and
    c. Creates an instance of the AbstractXML class, passing the device and course number.
8. The AbstractXML class does the following:
    a. Determines the stylesheet to use (by device language),
    b. Parses the selected stylesheet and input XML document to create DOM objects,
    c. Dynamically modifies the stylesheet with the course abstract to select,
    d. Invokes the LotusXSL processor (input the stylesheet and XML document DOM objects) creating a DOM object that represents the course abstract, and
    e. Returns the resulting DOM object to the Abstract servlet.
9. The Abstract servlet transforms the DOM Object into a print stream which is then sent back to the device (** end of a dialog ***).

Both the architecture picture (Figure 5) and the steps listed above show how the components and dialog interact with the device user. Before getting into the code, it is important to understand that some of the technology used is somewhat immature. Here are some of the considerations that were taken into account:

- The Apache Xerces XML Parser (version 1.0.3) and the LotusXSL (version 1.0.1) were used. The LotusXSL processor() method used accepts DOM objects for the .xml and .xsl files inputs and generates a DOM object. The other processor methods had problems dealing with the CDATA in the stylesheets. When the CDATA contained tags (<tag>), the other processor methods emitted the character encoding equivalent for the less than (<) and the greater than (>) signs which were not usable.
- Different browsers interpreted the incoming ASCII data stream differently. They sometimes misinterpreted the white space causing the browsers to have syntactically invalid URLs. Alternatives had to be created (shown in the *Small Device Course Selection* section).
- Emulators were used for the cell phones, palm pilots, and webTV, which may not depict current real device technology, but was adequate for testing a variety of devices.
- The WebSphere application server and Java components are both current release and PTF levels.
    - ❖ Caution was used when placing the .xml and .xsl files on the Integrated File System (IFS) to ensure the code looked in that directory path. The .xml and .xsl files were placed in the root directory to simplify the work.
    - ❖ Apache Xerces XML Parser, LotusXSL .jar files, and the CourseXML and AbstractXML classes (within the xmlproject directory) where put into the WebSphere system class path (the bootstrap.properties file) rather than the user path (servlets.property file). See Figure 6 for details.

```
java.classpath=/home/xmlproject:/home/gwhttp/lotusxsl.jar:/home/gwhttp
alan.jar:/home/gwhttp/xerces.jar:/QIBM/ProdData/NetCommerce/servlet/li
bedrock.jar:/QIBM/ProdData/IBMWebAS/web/classes:/QIBM/ProdData/HostPub
sher/common:/QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar
```

Figure 6 bootstrap.properties file

Application Implementation

This section discusses each of the components used to build the application. It is organized by how the application (while executing) invokes the components. Let's begin with the courses_xml.jsp.

courses_xml

The first step in the application dialog is to provide the appropriate 'home page' tailored to the device making the request. The home page must be in the right browser language (WML or HTML) and be sensitive to the device specifics (screen size). This can be done fairly easily in one of two ways. First, the automatic browser detection available on many HTTP servers (including the OS/400 web server V4R3 and later) can be used to determine the device. Secondly, this can be done dynamically in a Java based web server application using a servlet or Java Server Page (JSP).

A JSP is special form of web page that includes HTML for the display and the user interface, and Java (servlet) application code for integrating server processing and dynamic page generation. The JSP was chosen because it allows a solution that is HTTP server independent and it does not require a web server restart if support is added for other browser types. Figure 7 contains examples of the user agent string that the browser sends to the server, which can be used to ascertain the browser / device type.

```
Mozilla/4.0 (compatible; MSIE 5.0; Windows NT; DigExt) {Internet
Explorer 5 on an NT client pc, HTML}
Palmscape/PR5 (PalmPilot Pro:I) {IBM WorkPad/Palm Pilot browser, HTML}
WapIDE-SDK/2.0; (R320s (Arial)) {WAP browser on cellular telephone
device, WML}
```

Figure 7 user agent string

Our web application is rendered in three different formats:

1. Standard web page (desktop PC browsers) — displays HTML (for a big display), the pvc.gif graphic, and uses file courseselection.html
2. Smaller web page (Palm Pilot and WebTV) — displays HTML (for a small display), the roadie.gif graphic, and uses file courseselection_s.html
3. WML based pages for WAP browsers (cell phone device) — displays WML consisting of a deck of cards and uses file courseselection.wml.

In all cases, the client requests the web page courses_xml.jsp and the server logic determines the 'home page'. The first statement (in Figure 8) places the browser user agent into string variable userAgent. The next three statements define the default HTML page — comprised of a

header, body section, and footer for standard browsers.

```
<% String userAgent = request.getHeader("User-Agent");
  String header="<html><title>Road Warrior</title><body><center>";
  String body="<h1>Apps Central</h1><img src=\"pvc.gif\"
alt=\"Applications Menu\"><p><b>Choose Topic</b><p>
<a href=\"courseSelection.html\">Courses</a><p>
<a href=\"courseSelection.html\">Customers</a><p>
<a href=\"courseSelection.html\">Catalogs</a>";
```

Figure 8 initial jsp code

The code (shown in Figure 9) determines if the request is from a cell phone, by using the indexOf () method to check the user agent string for either “Wap” / “WAP”. If it is a cell phone, the output stream gets a WML card that briefly returns a Connecting.... message to the phone and then accesses the WML card deck contained in the courseselection.wml file. Note the MIME content type must be appropriately set. To better understand WML and WAP, see an overview in ***Appendix A*** of this document.

```
<%  if ((userAgent.indexOf("Wap") >= 0) | (userAgent.indexOf("WAP") >=
)) {
  header="<?xml version=\"1.0\"?>
  <!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"
“http://www.wapforum.org/DTD/wml_1.1.xml\">
  <wml>
  <card id=\"Apps_Central\" ontimer=\"courseselection.wml\"
title=\"Mobile Menu\" newcontext=\"true\">";
    body="<timer value=\"2\"></timer>
    <p>Connecting....</p>";
    footer="</card>
  </wml>";
    response.setContentType("text/vnd.wap.wml;charset-ISO-8859-1”);
```

Figure 9 WML connecting card

Notice (in Figure 9) the extensive use of the slash quote (\") characters to represent the quote (") character in the output data stream.

Figure 10 contains the JSP code sent to the page header, body and footer components appropriate for the device.

```
<% out.println(header); %>
<% out.println(body);  %>
<% out.println(footer); %>
<% out.close(); %>
```

Figure 10 JSP output statements

The courseSelection._ML files (which are output to the device) are in Appendix B. Figure 11 shows the JSP results displayed on the various devices. The devices are: WebTV (on the left side), a Palm Pilot (in the center), an Ericsson cell phone (on the right side).

Figure 11 Devices with Selection Display

The transmission back to the device completes the courses_xml.jsp components activity. The user enters the appropriate course selection criteria and submits the query to the server. In all

instances, the CourseSelection servlet receives the browser request, processes the request, and returns the results back to the requesting device.

CourseServlet

A servlet is a Java class that can receive inputs (such as an HTML form) from a web browser, perform any number of processing steps, and return appropriate output (such as HTML, graphics files, etc.) to the browser. Servlets run within a web application server environment and are analogous to traditional web server CGI programs. The WebSphere application server, in conjunction with the IBM HTTP Server for AS/400, provides the runtime environment for servlets and JSPs. For more information on servlets, see the reference section at the end of this document.

Figure 12 contains the import statements, class definition, and part of the doGet() method. The first three import statements are required for any servlet. The next six import statements are for the parser related APIs. Within the CourseServlet servlet definition, an instance of the PrintWriter is created to send data back to the browser. The doGet() method uses the getParameterValues() and getHeader() methods to get the selection criteria (day, topic, level) and device type. The servlet's HttpServletResponse object (called res) is assigned to the PrintWriter just created, which allows HTML or WML to be sent to the browser.

Figure 12 initial servlet code and start of doGet()

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Attr;
public class CourseServlet extends HttpServlet {
PrintWriter        out;

  public void doGet (HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
  {
   String day[] = req.getParameterValues("day");
   String topic[] = req.getParameterValues("topic");
   String level[] = req.getParameterValues("level");
```

In Figure 13, the servlet code (still inside the doGet() method) determines the device language by checking the userAgent and stores it in the device variable. This is very similar to the browser detection logic used in the JSP. The first if test checks for a Wap / WAP string in the userAgent indicating this is a cell phone requiring WML as output. If it is a cell phone, the appropriate output MIME type is set , two WML statements are put into the output stream, and the device variable is set to “WML”. The two WML statements put into the output stream are the xml version statement and a DOCTYPE statement indicating this is a WML document. Some cell phone browser requires these statements.

Warning: The XSL stylesheet used to create the output stream should create the two WML statements; however, the processor would not allow these two statements before the root <wml> tag in the output wml document. The only alternative was to create them here.

The second if test checks for a Palm Pilot or WebTV browser, sets the standard HTML output MIME type (required for HTML enabled small' devices), and sets the device variable to “LHTML”. For a request from other browsers, such as a PC, the output MIME type is set to HTML and the device variable is set to “BHTML”.

Figure 13 Device determination in doGet()

The try catch block, in Figure 14, contains the code to create an instance of the class (CourseXML) that performs the transformation. The CourseXML constructor creates an instance of the Course XML class and takes the course selection criteria and the device language as input. The CourseXML selectCourses() method is invoked which returns the resulting document as a DOM object. The document is then processed by the printElement() method.

```
if ((userAgent.indexOf("Wap")>= 0) | (userAgent.indexOf("WAP")>= 0 )) {

    res.setContentType("text/vnd.wap.wml; charset=ISO-8859-1");
   out.println("<?xml version=\"1.0\"?>");
   out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"
\"http://www.wapforum.org/DTD/wml_1.1.xml\">");
   device = "WML";
   }
else
 if ((userAgent.indexOf("Web")>= 0) | (userAgent.indexOf("Palm")>= 0)
 userserAgent.indexOf("Hand")>= 0)){
       res.setContentType("text/html; charset=ISO-8859-1");
        device = "LHTML";
        }
    else
```

```
try {
   CourseXML x = new CourseXML(topic[0],day[0],level[0],device);
   Document doc = x.selectCourses();
    if (doc != null){
     Element root = doc.getDocumentElement();
     printElement(root);
    }
  } catch (Exception e) {
      out.println("found exception calling CourseXML");
  }
```

Figure 14 try / catch block

The document that is returned from the selectCourses() method is then processed by the printElement() method. This method reads the DOM using DOM APIs to access the information in the document and generates output into the PrintWriter data stream which is sent back to the device. Details of this method are described in the white paper, ***Using the IBM XML Parser V2***, on the PartnerWorld for Developers, AS/400 website (See the reference section at the end of this paper.) Figure 15 contains the printElement() method code.

```
private void printElement(Element element){
      int k;
      NamedNodeMap attributes;
      NodeList children = element.getChildNodes();

      out.println("<" + element.getNodeName());
      attributes = element.getAttributes();
 if (attributes != null) {
      for (k=0; k < attributes.getLength(); k++) {
         out.println(" " + attributes.item(k).getNodeName());
             out.println("=\"" + attributes.item(k).getNodeValue()+
"\"");
      } // end for
 } // end if
 if (element.hasChildNodes()){
      out.println(">");
      for (k=0; k<children.getLength(); k++) {
       if(children.item(k).getNodeType() ==
org.w3c.dom.Node.ELEMENT_NODE){
             printElement((Element) children.item(k));
       }else if (children.item(k).getNodeType() ==
org.w3c.dom.Node.TEXT_NODE){
         out.println(children.item(k).getNodeValue());
       }// end else
   } // end for
   out.println("</"+element.getNodeName() + ">");
```

Figure 15 printElement()

Once the document is in output stream, the servlet is done processing. Now, let's look at the CourseXML java class and see what's happening when calling its methods.

CourseXML

CourseXML is a java class which has two methods: a constructor and the courseSelection() method. The CourseServlet class creates an instant of the CourseXML class passing the course of study, day, level (if appropriate), and the device as parameters. The CourseXML constructor is shown in Figure 16. The constructor creates instance variables for the parameters that were passed. It then determines the stylesheet to use placing the file name into the XSL_FILE object and converts the weekday into a number as defined in the XML document.

```
public CourseXML(String inT, String inDay, String inLevel, String inDe
)
{
 topic = inT;
 level = inLevel;
 device = inDev;
 if (inDev.equals("WML")) XSL_FILE = "toWML.xsl";
  else if (inDev.equals("BHTML")) XSL_FILE = "toBigHTML.xsl";
   else if (inDev.equals("LHTML")) XSL_FILE = "toLitHTML.xsl";
    else System.out.println("Bad Device");
 //* convert day of week to number
 if (inDay.equals("Monday")) day = "1";
  else if (inDay.equals("Tuesday")) day = "2";
   else if (inDay.equals("Wednesday")) day = "3";
```

Figure 16 CourseXML constructor

The CourseXML.selectCourses() method (shown on the next page in Figure 17) parses the selected stylesheet and the input XML document to create DOM objects for each of them. These documents do not contain a DOCTYPE so the parser runs in non-validating mode.

Next, an instance of the XSLProcessor is created. The selection criteria used within the stylesheet is dynamically modified by passing the topic, day and level information to the stylesheet, via the processor.setStylesheetParam() method. The processor.process() method is executed with both the XML document DOM and the stylesheet DOM as input. The processor.process() method creates a new DOM object according to the rules within the stylesheet. This new DOM object is returned to the CourseServlet.

```
public Document selectCourses(){
  Document docOut = null;
  if (XSL_FILE != null){
    //****  parse the XSL document & the XML document
  try {
   DOMParser parser = new DOMParser();
   parser.parse(XSL_FILE);
   Document xSSheet =  parser.getDocument();
   DOMParser parserX = new DOMParser();
   parserX.parse(XML_FILE);
   Document xXMLDoc =  parserX.getDocument();
   XSLProcessor processor = new XSLProcessor();
   processor.setStylesheetParam("sTopic",
       processor.createXString(topic));
   processor.setStylesheetParam("sDay",
processor.createXString(day));
   processor.setStylesheetParam("sLevel",
  processor.createXString(level));
     //** we are using this process method because it works
   docOut = processor.process(xXMLDoc, xSSheet, null);
 } catch (Exception e) {
           System.out.println("Exception in try block");
           e.printStackTrace(); }
```

Figure 17 CourseXML selectCourses()

CourseSelection Stylesheets

An eXtensible Stylesheet Language (XSL) stylesheet can contain either **template rules** for transforming an XML document into another markup language (_ML) including HTML, or **formatting objects** used to format an XML document for display in a browser. The XSL stylesheets used in this paper contain template rules to transform an XML document into another _ML. Both the stylesheet and the XML document are input to a processor, like LotusXSL, which converts the XML document content into another document type, according to the rules specified in the XSL stylesheet.

Stylesheet for Palm Devices and webTV

The XSL stylesheet, in Figure 18, filters and converts the course catalog XML document content into HTML (called little HTML) for a Palm device or webTV. XSL is an XML dialect, so line 1 specifies the XML version information and line 2 identifies this document as an XSL stylesheet and determines the name space and version used. Line 2 and all subsequent lines which start with the **<xsl:** syntax are XSL statements. Any content without the **<xsl:** designation is placed into the output stream, so the <HTML> tag, <H1> tag, <p> tag, text and so on are output data.
The stylesheet details are:

- Lines 3 - 5 — **<xsl:param** statements each accept a parameter passed by the processor.setStylesheetParam() method found in CourseXML selectCourses() method.
- Line 6 - 36 — template rule matching against the root (Courses tag) within the XML document. This single rule drives the creation of the output content.
- Lines 7 - 10 are considered output because they do not contain the <xsl: designation.
- Lines 11 - 13 are mixed with direct output and XSL statements. The **<xsl:value-of-select** statements (on line 11-13) output the parameters passed into this stylesheet (in lines 3 - 5).

- Line 20 is the start of a **<xsl:for each** block that matches against each Course within the XML document. For each Course within the course catalog, the processing inside the **<xsl:for each** block (starts on line 20 and ends on line 33) occurs.
- Lines 21 - 23 contain <xsl:if test statements that filter the courses looking for those courses that match the selection criteria (passed in as params).
- Lines 24 - 29 have specific element and attribute content that becomes part of the output by using the <xsl:value-of select statements. Each **<xsl:value-of select** statement contains the path to a specific element or attribute.

Figure 18 Palm Device XSL Stylesheet

- Lines 27 - 28 contain CDATA statements. The contents of these statements are directly output. An example of the output is shown in Figure 19.

```
<html>
<H1>Courses Available</H1>
<p> Your Selections<br/>
Day: 1<br/>                          // selection criteria
Topic: XML<br/>                      // selection criteria
Level: Introductory<br/>             // selection criteria
</p>
<p>Click title for details<br/></p>
<table>
<tr><td>Time</td><td>Course #</td><td>Title</td></tr>  // table headin
<tr><td>9:00AM</td><td>X101</td>
<td><a href="http://myserver/Servlet/AbstractServlet?X101"/>Basics of
XML</td></tr>
<tr><td>10:00AM</td><td>X102</td>
<td><a href="http://myserver/Servlet/AbstractServlet?X102"/>Fundamenta
of creating XML</td></tr>
```

Figure 19 output little HTML

Figure 20 shows the results displayed on a Palm Pilot and a webTV respectively.

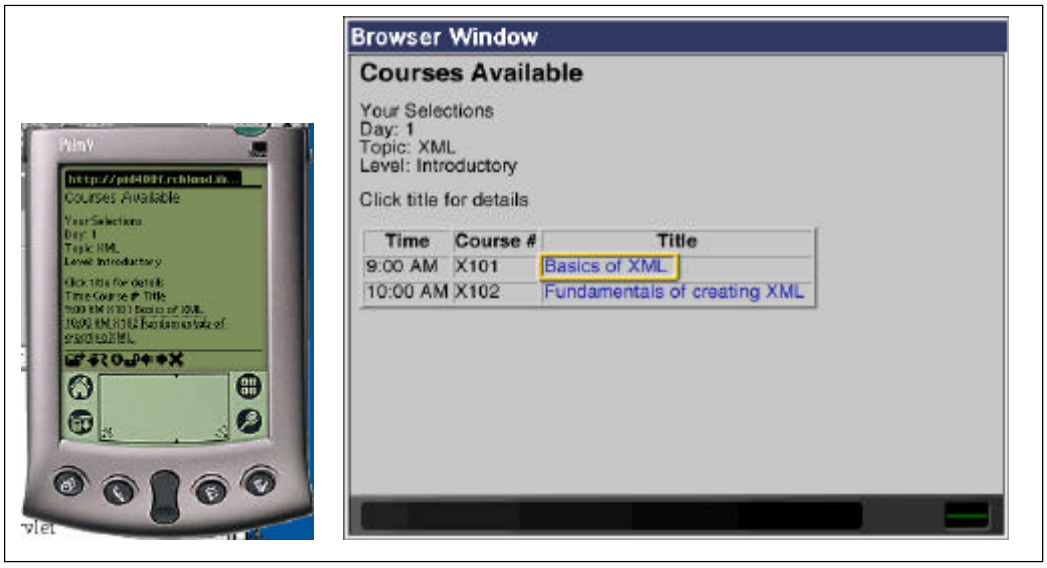


Figure 20 Palm Pilot and WebTV

Cell Phone (WML)

As shown in Figure 21, the XSL stylesheet is used to emit WML, which is accepted by the Ericsson cell phone used in this application. A WML document consists of one or more cards which make up the user dialog.

The WML document contains a single card. The <card> tag contains three attributes. The first attribute is the id which is the card name. The second attribute is the ontimer which automatically loads a WML script used for local text processing during page startup. Thirdly, the title attribute contains the text displayed on the cell phone. A timer statement triggers the ontimer actions init() method call. The statements within the template rule are very close to those discussed in the “Stylesheet for Palm Devices and webTV” section above. Because the display space is very small, limited content is output. The CDATA statements are used to construct the <option statements, which allow the user to select more information about a particular course. Both the CourseDetails/CourseID content and the courseDetails/ShortTitle content are included within the <option statement.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:param name="sTopic" select="Java"/>
<xsl:param name="sDay" select="1"/>
<xsl:param name="sLevel" select="Intermediate"/>
<xsl:template match="Courses">
 <wml>
 <card id="results"
ontimer="http://myserver.com/home/weave/abstract.wmls#init()"
title="Selections">
 <timer value="1"/><do type="prev"><prev></prev></do>
<p> Criteria:<br/>
 <xsl:value-of select="$sTopic"/>,
 <xsl:value-of select="$sDay"/>,
 <xsl:value-of select="$sLevel"/> <br/>
 <b>Select title for details </b>  <br/>
 <select name="coursenumber">
 <xsl:for-each select="Course">
 <xsl:if test="@TopicArea=$sTopic">
 <xsl:if test="CourseDetails/Level=$sLevel">
  <xsl:if test="Schedule/Occurance/Day=$sDay">
  <![CDATA[<option value="]]><xsl:value-of
select="CourseDetails/CourseID"/><![CDATA["
onpick="http://myserver.com/home/weave/abstract.wmls#trimmer()"> ]]>
  <xsl:value-of select="CourseDetails/ShortTitle"/>
<![CDATA[</option>]]>
  </xsl:if>
 </xsl:if>
 </xsl:if>
 </xsl:for-each>
</select></p></card></wml>
```

Figure 21 Cell phone XSL Style sheet

Figure 22 contains the resulting WML and the WML displayed on a cell phone.

Figure 22 Resulting WML and Cell phone

Full Screen Browser

```
<wml>
<card id="results"
ontimer="http://myserver.ibm.com/home/weave/abstract.
ls#init()" title="Selections">
<timer value="1"/>
<do type="prev"><prev/></do>
<p>
Criteria:<br/>
XML, 1, Introductory<br/>
<b>Select title for details </b>
<br/><select name="coursenumber">
<option value="X101"
onpick="http://myserver.com/home/weave/abstract.wmls#
immer()">
Basis XML </option>
<option value="X102"
onpick="http://myserver.com/home/weave/abstract.wmls#
immer()">
Create XML </option>
</select>
</p>
</card>
</wml>
```

The XSL stylesheet, in Figure 23, transforms the XML document into HTML for a full screen browser environment and emits more course information because of the additional display space available. This example shows how stylesheets can be used to customize the output generated from a single XML document. There are three template rules in this stylesheet. Let's look at each of them.

The first template rule**, <xsl:template match="Courses">**, matches against the root of the XML document and creates the general browser output (structural tags, heading tags, table headings tags, and some text). Only the topic and day values are used for selection criteria. The table has more cells than previously seen, including Level, Start Time (Duration), Course#, Title, Presenter, and Location. The <xsl:apply-templates> statement tells the processor to look for more rules to process before outputting the rest of the HTML document.

The second template rule, **<xsl:template match="Course" >**, matches against the Course elements within the XML document. It contains two <xsl:if test statements for the Topic and Day which is the users selection criteria. The rule also contains a <xsl:choose> statement where the <xsl:when clause matches against the Level elements content converting from the word (Introductory, Intermediate, Advanced) to a numeric equivalent (1, 2, 3 respectively). The rules <xsl:apply-templates statement tells the processor to look for another template rule to execute

on the specified element path. The seven <xsl:value-of select statements obtain other content from the XML document which becomes part of the table content.

The third template rule, **<xsl:template match="Schedule/Occurance/Time" >**, matches on the Time element which is found by its parentage (Schedule/Occurance/Time). The Time elements @Period attribute value is output.

Figure 23 Browser XSL Style sheet

The resulting HTML is shown in Figure 24 and the browser display is shown in Figure 25.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:param name="sTopic" select="Java"/> <xsl:param name="sDay"
select="1"/>
<xsl:param name="sLevel" select="Intermediate"/>
<xsl:template match="Courses">
 <html>  <H2><i>Courses Available</i></H2>
 <H3><i>by Topic = <xsl:value-of select="$sTopic"/> and Day =
<xsl:value-of select="$sDay"/></i></H3>
 <body Text="#0009FF"> <table border="2">
 <tr><th>Level*</th><th>StartTime (Duration)</th><th>Course#</th>
 <th>Title</th><th>Presenter</th><th colspan="2">Location</th></tr>
 <xsl:apply-templates> </xsl:apply-templates>
 </table>  </body>  <h3><i>* Legend</i></h3>
 <UL><li>1 = Introductory</li>  <li>2 = Intermediate</li>  <li>3 =
Advanced</li> </UL>
 </html>
</xsl:template>
<xsl:template match="Course" >
 <xsl:if test="@TopicArea=$sTopic">
 <xsl:if test="Schedule/Occurance/Day=$sDay">
  <tr> <td Valign="top" Align="center">
    <xsl:choose>
    <xsl:when test="CourseDetails/Level='Introductory'"><b> 1
</b></xsl:when>
     <xsl:when test="CourseDetails/Level='Intermediate'"><b> 2
</b></xsl:when>
     <xsl:when test="CourseDetails/Level='Advanced'"><b> 3
</b></xsl:when>
    </xsl:choose>    </td>
  <td Valign="top"> <xsl:apply-templates
select="Schedule/Occurance/Time"/>
   (<xsl:value-of select="CourseDetails/Duration"/>)</td>
  <td Valign="top" ><xsl:value-of
select="CourseDetails/CourseID"/></td>
  <td><i><b><xsl:value-of select="CourseDetails/LongTitle"/></b> </i>
   <p><xsl:value-of select="CourseDetails/Abstract"/></p> </td>
  <td Valign="top"><xsl:value-of
select="CourseDetails/Presenter/Name"/></td>
  <td Valign="top">Bldg: <xsl:value-of
select="Schedule/Occurance/Location/Building"/>
```

Figure 24 PC desktop Browser HTML

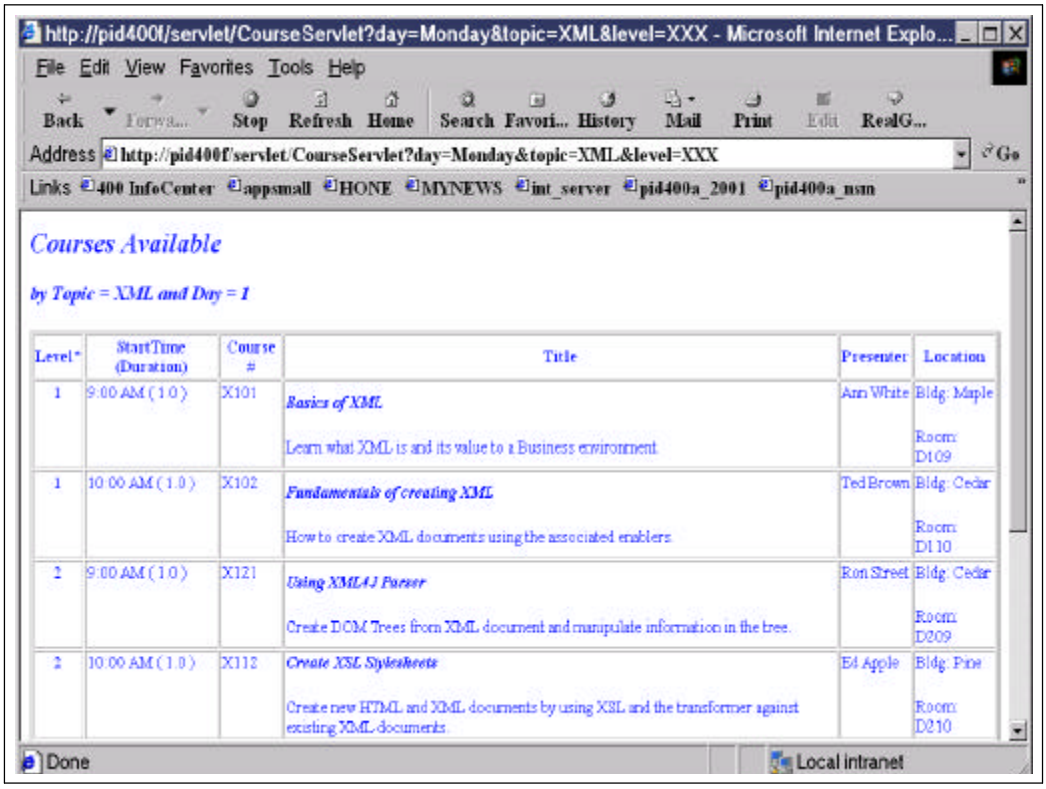
Figure 25 PC desktop Browser

This concludes the dialog for the PC desktop browser; however, the small devices can continue the selection process and get an abstract for one of the courses that were returned.

```
<html>
<H2><i>Courses Available</i></H2>
<H3><i>by Topic = XML and Day = 1</i></H3>
<body Text="#1E90FF">
<table border="2">
<tr><th>Level*</th><th>StartTime (Duration)</th><th>Course #</th>
 <th>Title</th><th>Presenter</th><th colspan="2">Location</th></tr>

<tr><td Align="center" Valign="top"><b> 1 </b></td><td
Valign="top">9:00AM(1.0) </td>
  <td Valign="top">X101</td><td><i><b>Basics of XML</b></i>
<p>Learn what XML is and its value to a Business environment.</p></td>
<td Valign="top">Ann White</td>
<td Valign="top">Bldg:Maple<P>Room:D109</P></td></tr>
table entries for course# X102, X121, X112, X821, X812
</table>
</body>
<h3><i>* Legend</i></h3>
<UL><li>1 = Introductory</li>
<li>2 = Intermediate</li>
```

Small Device Course Selection



For smaller devices, a list of courses meeting the search criteria was returned to the user. The user has the opportunity to get a course description by selecting a specific course. The application uses the same techniques (defined in the CourseServlet section of this paper) to retrieve a course description which is tailored to the device/browser environment.

Figure 22 and Figure 24 (above) contains the WML and HTML sent to the small browsers. For the small HTML devices (if the user selects course X102), the submit URL is **http://myserver.com/servlet/AbstractServlet?course=X102**. This sends the course number to the AbstractServlet servlet, which processes the request and returns the appropriate information wrapped in HTML. For the cell phone, the

```
String course[] = req.getParameterValues("course");
 String course1 = course[0].trim();
 String userAgent = req.getHeader("User-Agent");
 String device = null;
```

http://myserver.com/home/weave/abstract.wmls#trimmer() submit URL calls a WML script routine. This script can access the course number selection variable creating a valid URL which invokes the AbstractServlet servlet. Figure 26 contains the script code.

Figure 26 WML script code

The WML script routine trims all leading and trailing spaces from the variable (selected course number), concatenates that variable to the rest of the URL, thereby creating a valid URL. This work around was required for the WAP browser environment because the cell phone browser interprets the URL in the data stream (href="http://myserver.com/servlet/AbstractServlet?course= X101") as syntactically invalid. Also, one of the Palm Pilot browsers exhibited this behavior. The trimmer() function is a WML script common library function. Once the URL (http://myserver.com/servlet/AbstractServlet?course=X101) is constructed, the AbstractServlet is invoked.

```
extern function trimmer() {
var instring= WMLBrowser.getVar("coursenumber");
var trimmedstring = String.trim(instring);
var gourl =
"http://myserver.com/servlet/AbstractServlet?course="+trimmedstring;
 WMLBrowser.go(gourl);
```

AbstractServlet

The AbstractServlet code is similar to the code for the CourseServlet, which was discussed earlier. The package import statements and the class definition are very similar and need not be revisited. This servlet receives only one input parameter (course number). The code uses the req.getParameterValues() method (in Figure 27) to access the course number. Another String object (course1) contains the results of using the trim() function on the course String object. This was done because, in some circumstances, the small device browsers were sending escaped values to the servlet like, %20X101 or X101%20. Trimming these white spaces was necessary because the stylesheet needed the exact course number variable to match against the CourseID in the XML document.

Figure 27 Processing Strings

The AbstractServlet invokes (as shown in Figure 28) the AbstractXML constructor (which creates the requested data) passed to it. The AbstractServlet invokes the getAbstract() method and returns the resulting DOM object. The AbstractServlet's printElement() method (identical to the CourseServlet printElement() method) generates the output to the device.

Figure 28 Invoking AbstractXML

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:param name="course" select="XXXX"/>
<xsl:template match="Courses">
<html>
<H1>Course Abstract: <xsl:value-of select="$course"/> </H1>
<p>
 <xsl:for-each select="Course">
  <xsl:if test="CourseDetails/CourseID=$course">
    <xsl:value-of select="CourseDetails/Abstract"/>
  </xsl:if>
 </xsl:for-each>
</p>
</html>
</xsl:template>
</xsl:stylesheet>
```

AbstractXML

The AbstractXML class, like the CourseXML class, uses both the parser and LotusXSL processor. The AbstractXML constructor expects two parameters: the courseId and the device language. The getAbstract() method works like the CourseXML selectCourses() method discussed earlier. The stylesheets, in Figure 29 and Figure 30, filter and transform the XML.

```
try {
```

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:param name="course" select="XXXX"/>
<xsl:template match="Courses">
 <wml>
 <card id="abstract" title="Abstract">
 <do type="prev"><prev></prev></do>
 <p>
  Course:<xsl:value-of select="$course"/>
 </p><p>
  <xsl:for-each select="Course">
  <xsl:if test="CourseDetails/CourseID=$course">
    <xsl:value-of select="CourseDetails/Abstract"/>
  </xsl:if>
   </xsl:for-each>
</p></card></wml>
```

Figure 29 Style sheet for HTML

Figure 30 Style sheet for WML

Figure 31 shows the results on a Palm Pilot and a cell phone.

Figure 31 Course Abstract displayed

In Conclusion

In this paper, a number of technologies were covered — XML, XML enablers, Web application servers, Java, servlets pervasive computing devices, etc. The intent was to show how these technologies can be used with AS/400 to provide an application solution. Providing web access to business applications and data from emerging devices will continue to grow in importance. Creating multiple versions of web sites or web applications formatted and tagged for different device types is an expensive, inflexible, difficult to manage solution.

AS/400 can be used, today, to build solutions for these new business challenges and opportunities. The WebSphere Application Server, Java Virtual Machine (JVM), HTTP server, parser, and XSL processor all run natively on AS/400 and are available at no extra charge. Hopefully, this paper has shown how implementing these new technologies is possible and that they can be used to build new business solutions for users and customers.

For Additional Information

XML Papers and Links

- PartnerWorld for Developers, AS/400
  http://www.as400.ibm.com/developer/java/xml/index.html
- IBM XML
  http://www.ibm.com/developer/xml/

General AS/400 and IBM

- AS/400 Java web site
  http://www.as400.ibm.com/java
- AS/400 Websphere website http://www.as400.ibm.com/products/websphere
- IBM Pervasive Computing web site
  http://www.ibm.com/pvc
- AS/400 PartnerWorld Pervasive Computing web site
  http://www.as400.ibm.com/ebiz/handheld

Redbooks (available at http://www.redbooks.ibm.com)

- Building AS/400 Applications for IBM WebSphere Standard Edition 2.0 SG24-5635-00
- Building AS/400 Appls for WebSphere Advanced 3.0 SG24-5691-00
- Web Enabling AS/400 Applications with IBM WebSphere Studio SG24-5634-00

Wireless, WAP and WML References

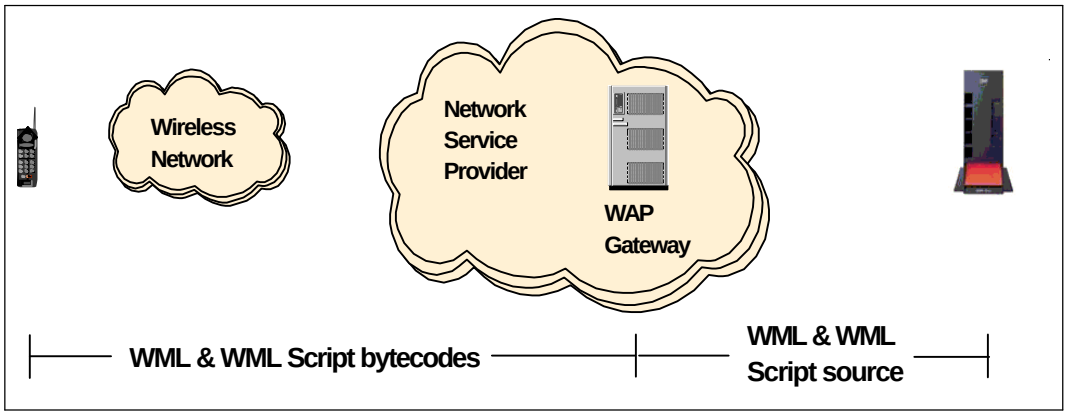
- http://www.wap.net
- http://www.wapforum.org
- http://www.cdpd.org
- http://www.gsmworld.com

Appendix A: WML and WAP

Perhaps the most important consideration in developing and deploying web based applications for the cellular telephone devices is that they do not utilize HTML. Rather, they utilize an XML oriented relative to HTML called Wireless Markup Language (WML). Since cell phones are designed primarily for voice applications and long battery life, the local processing resources tend to be minimal. Nevertheless, the WML environment also provides WML Script for enabling local processing capability. This is analogous to using JavaScript in an HTML browser environment for doing simple field validation checks, text processing, or numeric calculations. Here is the WML source used to build the card shown in the cell phone display:

```
?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
           "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="homepage" title="WML Example" >
  <p>AS/400 Homepage Link<br/>
   <a id="400link" href="http://209.163.29.221/courses_xml.jsp"
title="Click Here">
   </a>
  </p>
  </card>
```

In the main section of the paper, how the Wireless Application Protocol (WAP) utilizes a gateway between the wireless device and content host(s) was discussed. This is better depicted in a diagram.



The WML and WML Script source can be generated or natively stored directly on the server. However, the WAP protocol dictates that the cellular phone device receives the WML data in compiled byte code content to reduce the amount of data transferred and improve response time. The WAP gateway provides this function and acts as an application proxy to the cellular phone device. It can receive WML source content, compile it to byte codes, and deliver it to the WAP micro browser device. It also receives browser requests as compiled WML byte code, translates it to WML URL requests, and delivers it to the content or application server.

Appendix B: CourseSelection files

In this application, a Java Server Page (JSP) was used to determine the browser/device type and then deliver an appropriate 'homepage'. In this section, the document source is listed. The HTML source is fairly trivial; however, the WML source is a bit more involved.

courseselection.html (big HTML device type)

```
<html>
<title>Course Selection</title>
<center><h1>Enter Your Selection Criteria</h1></center>
<form name=input action="http://myserver.com/servlet/CourseServlet"
    <p><bold>Choose Day</bold>
    <select name="day">
      <option>Monday
      <option>Tuesday
    </select>
    <p><bold>Choose Topic</bold>
    <select name="topic">
      <option>XML
      <option>Java
    </select>
    <input type="hidden" name="level" value="XXX"> <br>
 <p> <input type=submit value="Find Courses">
   <input type=reset value="Reset">
  </form>
</html>
```

courseselection_s.html (small HTML device type)

```
<html>
<title>Course Selection</title>
<center><h1>Enter Your Selection Criteria</h1></center>
<form name=input action="http://myserver.com/servlet/CourseServlet"
    <p><bold>Choose Day</bold>
    <select name="day">
      <option>Monday
      <option>Tuesday
    </select>
    <p><bold>Choose Topic</bold>
    <select name="topic">
      <option>XML
      <option>Java
    </select>
   <p><bold>Chooose Level</bold>
 <select name="level">
      <option>Introductory
      <option>Intermediate
      <option>Advanced
    </select> <br>
 <p> <input type=submit value="Find Courses">
   <input type=reset value="Reset">
  </form>
</html>
```

courseselection.wml (WML device type)

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
           "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="intro" ontimer="#menu" title="Road Warrior" newcontext="true">
    <timer value="20"/>
    <p>
    <img width="20" hspace="1" src="roadie.gif" height="25" alt="Course
Schedule" vspace="1"/>
    </p>
    <p align="center">
    <b>Apps Central</b>
    <br/>
    </p>
</card>
<card id="menu" title="Applications" newcontext="true">
    <do type="prev">
      <go href="#intro"></go>
    </do>
    <p><strong>Choose Topic</strong>
    <br/>
    <select name="application">
      <option value="CRS" onpick="#coursemenu">Courses
      </option>
      <option value="CST" onpick="#coursemenu">Customers
      </option>
      <option value="CTG" onpick="#coursemenu">Catalogs
      </option>
    </select>
    </p>
  </card>
  <card id="coursemenu" title="Course Schedule" newcontext="true">
   <do type="prev">
      <go href="#menu"></go>
    </do>
    <p><strong>Choose Day</strong>
    <br/>
    <select name="day">
      <option value="Monday" onpick="#topic">Monday
      </option>
      <option value="Tuesday" onpick="#topic">Tuesday
      </option>
    </select>
    </p>
  </card>
<card id="topic" title="Course Topics">
   <do type="prev">
      <go href="#coursemenu"></go>
    </do>
    <p><strong>Choose Topic</strong>
    <br/>
    <select name="topic">
      <option value="Java" onpick="#level">Java
```

```
      </option>
      <option value="XML" onpick="#level">XML
      </option>
    </select>
    </p>
  </card>
<card id="level" title="Course Levels">
   <do type="prev">
      <go href="#topic"></go>
    </do>
    <p><strong>Choose Level</strong>
    <br/>
    <select name="level">

      <option value="Introductory" onpick="#results">Introductory
      </option>
      <option value="Intermediate" onpick="#results">Intermediate
      </option>
      <option value="Advanced" onpick="#results">Advanced
      </option>
    </select>
    </p>
  </card>
  <card id="results" title="Search"><p>
<do type="prev">
      <go href="#level">
      </go>
    </do>
    <do type="accept">
<go href="http://myserver.com/servlet/CourseServlet">
    <postfield name="day" value="$day"/>
      <postfield name="topic" value="$topic"/>
      <postfield name="level" value="$level"/></go></do>
Day: $day<br/>
Topic: $topic<br/>
Level: $level<br/>
Submit?</p>
  </card>
</wml>
```

Trademarks

This publication may have referred to products that are not available in your country.

IBM, AS/400, OS/400, OS/2, C/400, 400, Client Access/400, VisualAge, are registered trademarks of the IBM Corporation in the United States or other countries or both.

Java, JDK, 100% Pure Java, JavaBeans, SUN, Solaris, Write Once, Run Anywhere and all Java-based trademarks and logos are registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Microsoft, Windows and the Windows logo are trademarks or registered trademarks of Microsoft Corporation.

IBM's VisualAge products and services are not associated with or sponsored by Visual Edge Software, Ltd.

IBM makes no commitment to make available any products referred to herein.

All other trademarks and registered trademarks are the properties of their respective companies.

For More Information

About the Author

LindaMay Patterson PartnerWorld for Developers, AS/400 Advisory Software Engineer

LindaMay Patterson is an Advisory Programming in PartnerWorld for Developers, AS/400 at IBM Rochester. She has worked at IBM for 24 years in various Business Application environments. Currently, she is part of the PartnerWorld for Developers, AS/400 Java team working with Enterprise JavaBeans. Prior to this, she worked on the SanFrancisco product developing the education packages and working on SanFrancisco in Germany helping to define the product content. Her Application background is primarily in Distribution and Logistics Systems.

George Weaver PartnerWorld for Developers, AS/400 eBusiness consultant

George Weaver is an eBusiness consultant with the PartnerWorld for Developers, AS/400 organization at IBM in Rochester, Minnesota. He has been with IBM for 18 years, with the last 7 in the PartnerWorld for Developers, AS/400 organization providing consultation, education and writings to help customers and business partners enhance their AS/400 applications and services portfolios. His current areas of focus include AS/400 networking and communications, mail and messaging, HTTP server, directory, TCP protocols, and Lotus Domino for AS/400 topics.

For more information about PartnerWorld for Developers, AS/400

To get more details on becoming a member of PartnerWorld for Developers and AS/400, visit our Web site at: http://www.ibm.com/as400/developer/membership/reg_info.html

Check out our PartnerWorld for Developers, AS/400 service offerings at: http://www.ibm.com/as400/developer/java

Production

Teresa Powell PartnerWorld for Developers, AS/400 Member Services