DEVELOPING DYNAMIC WAP-ENABLED WEBSITES USING ORACLE'S XML TOOLS

Doug Burns, Independent

INTRODUCTION

In spite of the continuing debate about the technical merits and user demand for the Wireless Application Protocol (WAP), there is general agreement that the number of users of mobile internet access devices world-wide is likely to exceed the number of users of desktop devices within the next few years.

This paper explains how to use Oracle's XML Developer's Kit (XDK) to develop dynamic data-driven web sites that can be accessed by users from a variety of different handheld devices. I will concentrate on the Wireless Application Protocol (WAP) because it is the most unusual platform to implement. However, by using eXtensible Mark-up Language (XML) as the underlying data format and the eXtensible Stylesheet Language (XSL) to describe presentation formatting we can simplify the implementation of any new device formats that you need to support.

In the current development climate of multiple presentation environments and short project timescales, the power, flexibility and implementation speed available to the developer who uses Oracle's XML tools are incredibly useful in overcoming these challenges.

WHY WAP?

There seems little argument about the increasing demand for mobile data access, but many commentators feel that WAP is likely to be superseded within a couple of years by information delivery protocols that will be a better match for the higher-bandwidth wireless devices and communication protocols of the future. At the other extreme, some commentators feel that older facilities such as Short Messaging Service (SMS) offer sufficient functionality for current user demands. So let me explain a few reasons why I feel that WAP services are worth implementing.

- WAP is here now.

  I own and use a WAP-enabled mobile telephone and so do my friends. Manufacturers (at least in the UK and other parts of Europe) are subsidising the cost of new WAP-enabled telephones to encourage consumer sales and WAP-enabling all of their most up-to-date handsets. Online banks are giving away WAP-enabled phones to encourage users to sign-up for their services. The result is that many users are acquiring WAP phones without making any decisions at all about which is the best protocol. Even if they decide to analyse the various future protocols and delivery options – what are they going to do for the next year or two whilst they wait for these to become available? They want a new phone now and, whilst they might only use WAP services occasionally, there will still be far more potential users of WAP phones than other wireless access protocols for the foreseeable future.

- WAP has big vendor support.

  Maybe I'm a traditionalist, but I pay attention when organisations such as Nokia, Motorola, AT&T, BT and Ericcson, commit themselves to the development of a protocol such as

WAP. In my opinion, they understand consumer needs and current technical issues more clearly than the critics of WAP and have only been prepared to compromise on the functionality of the protocol itself in order to deliver something now.

- WAP is just one delivery option.

  This is the best bit. Even if I'm completely off the mark in my assessment of the current situation, the development approach described in this paper ensures that data presentation and delivery format are separated from the underlying data and application logic. That means if (when) another protocol becomes more effective and widely used, the extra implementation effort required would be limited. We can offer people information in the formats that they want; when they want it, rather than the formats that we deem technically sound; when they become available.

There are many more opinions about the pros and cons of WAP at the various WAP development sites listed in the Resources section at the end of the paper, but for the rest of the paper, I'll concentrate on one approach to the development work itself.

WML – The Wireless Markup Language

The mark-up language defined by the WAP protocol is WML. A full description and tutorial is beyond the scope of this paper, but I've found it an extremely simple language to use and I have included links to the WAP standards documentation and some of the better tutorials in the Resources section. I'll limit myself here to a very brief overview as an introduction to the small WML examples that we'll be working on.

Example 1 A WML User Input Form

```
<wml>
      <card id="searchString">

            <do type="accept">
                  <go method="post"
            href="http://localhost/xsql/jobserve.xsql">
                  <postfield name="mySearch"  value="$mySearch"/>
                  </go>
            </do>

            <p>
            Search String: <input name="mySearch"/>
            </p>
      </card>
</wml>
```

At first glance, WML looks like HTML and other mark-up languages. Note the <wml> and </wml> tags that enclose the whole document.

The most important difference between WML and HTML is that a WML 'page' is organised as a *deck* of *cards*, where each card represents one small screen-full of information. The entire deck is downloaded to the user's device when the request is received and the user can then navigate between the various cards using the card id, without accessing the server (unless, of course, the user requests another deck). This reduces the number of trips between server and client and allows some processing to take place within the client device (particularly when combined with WML Script – and ECMA Script derivative). The deck in example 1 only contains one card, surrounded by the <card> and </card> tags, but we'll work with examples later that have multiple, linked cards in one deck.

The <do> tag assigns an action to the 'accept' button on the mobile device. When this button is pressed, the post method will be used to submit a request to the web server on my local machine for the jobserve.xsql page. The value for the postfield is appended to the request and is requested from the user, using the <input> tag below. Note that, because the input tag does not have a matching closing tag, </input>, then it must end with a /.

This highlights another key difference between WML and most versions of HTML. It is a valid XML language. In other words it was written using XML so WML is defined by an XML Document Type Definition (DTD). The first noticeable effect of this tends to be an increase in the number of errors generated when you first start writing WML pages, because it is very particular about ensuring that opening tags have matching closing tags and that tags and their attributes are also case-sensitive.

Example 1 will be used by as the opening page of our prototype to allow the user to submit a user-defined query to our database. As we encounter other examples of WML, I'll explain how they work as a gentle introduction to the language itself. But, interestingly, a large part of our work will be performed using a familiar language, SQL, and a more recent development, XML.

XML AND THE ORACLE XML DEVELOPER'S KIT

Living in the uncertain development environment created by rapid implementation of new devices, applications and file formats, it's a relief to witness the growing acceptance of XML as the data exchange format of the future. There is a wealth of information available on the internet concerning XML and I have included some key links in the Resources section, but here are a few of the many positive attributes of XML that I think will prove particularly important

- ***It's a W3C standard*** and offers the possibility that everyone will be able to communicate more effectively over the internet in future by providing a rigorous, well-defined but extensible model to build new dialects. It sometimes helps me to think of XML as a kind of TCP/IP, HTTP or SQL for textual data, with a similar potential to accelerate the benefits that we extract from our use of computers.
- ***It makes meaning more transparent*** so that mobile agents will be able to operate in a more structured arena and make more intelligent decisions. As a veteran of data extract and load routines, it even offers certain benefits for that type of application.
- ***It separates presentation from data*** which will become more important as alternative access methods are used by users, depending on their location and needs. Despite the best efforts of previous mark-up languages their limitations have led to the need for more and more proprietary extensions.

The last attribute is of most interest to developers of web sites that need to deliver data in multiple formats. By using XML and XSL in combination, we are afforded a form of insurance against future browser developments; are able to implement multiple presentation formats for the same data; and allow our users more flexibility in how their data is formatted, without changing the underlying code that retrieves the data.

By developing our services so that all our content is generated and stored in XML format, we can delay decisions about the device presentation aspects until immediately prior to delivery. As the standards change and develop, we modify or add stylesheets that the XSL Transformation processor uses to generate the final content without affecting our underlying database access code. There will also be opportunities in future to offload some of the processing load used to render the output to the client device as XSL transformation facilities are built in to client software.

THE ORACLE XML DEVELOPER'S KIT

Oracle packages it's various XML development tools and Utilities as the Oracle XML Developer's Kit (XDK) which consists of the following tools, all of which can be installed inside an Oracle 8i database.

- XML Parser Java, PL/SQL, C++ and C versions available.

  The most important tool in many ways is the XML parser, because it is this on which the other tools are based. It supports XML 1.0 and can be used as either a validating or non-validating parser and supports both the Simple API for XML (SAX) or Document Object Model (DOM) approaches to parsing.

- XML Class generator Java and C++ versions available

  The XML Class generator can create source files containing class information based on a DTD so that different development teams can share class libraries that represent an agreed DTD specification.

- XML SQL Utility

  The XML SQL Utility is a package of Java classes that can be used to perform two complementary tasks :-

  Generate an XML file and associated DTD from a database query

  Load XML data into database tables

- XSQL Servlet

  The tool that we'll be concentrating on is the XSQL Servlet. This is a Java servlet which is added to your web server configuration using an appropriate Servlet engine. For example, I used Apache and the Jserv module to develop the examples included in this paper, but the excellent release notes give configuration instructions for a variety of common servers.

  When the user submits a request to the server for an *XSQL Page* (which is identified by the .xsql extension) it passes the request off to the servlet which parses the page and forwards any SQL queries to the database. When the results are returned, it converts them to XML format and, optionally, will invoke a built-in XSL Transformation processor to render the output based on an XSQL stylesheet specified in the XSQL page. We'll see a couple of examples of XSQL pages shortly.

At the time of writing, this toolkit is a free download from the XML software section at Technet (http://technet.oracle.com/tech/xml), but my understanding is that it will be incorporated in all future releases of the Oracle server. So all of this functionality comes for free!

I recommend that you download the XSQL Servlet distribution, which includes the XSQL Servlet that we will be using, as well as the XML SQL Utility and the XML Parser for Java on which it is based. The Release notes that are included are excellent and include detailed instructions to help you set up the servlet in a variety of common web server environments or use the lightweight Web-to-Go server that is supplied as part of the package. You might also want to refer to a previous version of this paper that focussed on installation issues at http://doug.burns.tripod.com/xml.html)

AN EXAMPLE APPLICATION

I developed a simple prototype for a WAP-enabled web-site recently over a couple of weekends, including designing and populating the database and installing and configuring the XSQL servlet, which gives an indication of how simple the tools are to use.

This prototype screen is the search facility for a IT recruitment web site that would store a candidate's search criteria and would allow the user to view the most recent job postings of interest. An essential aspect of the application is that up to date data would need to be retrieved dynamically. This would allow an IT consultant operating in the very dynamic UK consultancy market to look for work whilst enjoying their vacation on the beach. So, at the very least, even if the project was abandoned, I'd be able to use the application personally!

A critical aspect of design for mobile devices is that, because screen real estate and bandwidth are so limited on a mobile phone, the presentation should be basic, easy to navigate and concentrate on the key information. This has been raised as a criticism of WAP, but I suggest that there are certain applications where a basic user interface is an advantage.

A BRIEF INTERLUDE – UTL_HTTP AND DATA ACQUISITION

To get the site up and running in the shortest available time I would need to have access to a database of live job requirements. The recruitment site most commonly used by UK contractors is http://www.jobserve.com for it's large number of requirements and it's elegant design that uses the minimum number of extraneous frills. The site offers a daily comma-delimited text file containing all of the job requirements posted that day and I used SQL*Loader to load this in the early stages of the project.

The only problem with relying on this file was that the requirements might be up to a day old, which would defeat some of the purpose of using a mobile service – instantaneous access to current data. Then I remembered that Jobserve has a dynamically updated page containing requirements posted today, with the most recently posted listed first, at http://www.jobserve.com/asp/instants.asp. By using the UTL_HTTP.REQUEST_PIECES function supplied with the Oracle server to read this page, I would be able to update the data contained in the database in near real time and provide up to date information on demand. Some people call this screen scraping; some people call this data theft; I call it creative prototyping, but I would not recommend that others take this approach! (Despite the fact that it proved very effective.)

Interestingly, because the job_id is included within the link on the Instants page, I was able to establish that each job was unique and then submit an additional request for the job details screen, passing the job_id as a parameter. Then all of the job, agency and agent details could be gathered and loaded into the database.

There is insufficient space to include a full listing of the PL/SQL Package I developed to do this, but it's an interesting example of how useful UTL_HTTP can be. If you are interested, send me an email at the address listed at the end of the paper and I'll be happy to send you a copy.

The resulting database contained a number of tables containing information about jobs, candidates, recruitment agencies and individual agents, but our examples will be limited to the simple JOB table

```
JOB   (Subset of columns)
```

```
JOB_ID      NUMBER            NOT NULL
TITLE       VARCHAR2(40       NOT NULL
LOCATION    VARCHAR2(80)
SKILLS      VARCHAR2(255)
```

OUR FIRST XSQL PAGE

The first prototype screen contains the five most recently posted jobs that match the user's search criteria. The value for the search string would be supplied when the user completes the WML form listed in example 1, although it could also be stored in one of a catalogue of user search profiles stored in the database in a real system. The screen only needs to access one table in the database – the JOB table and present an initial WML deck that contains a top level card with each job title forming a hyperlink pointing to a related card containing further details about the job. This allows the user to have a quick view of the jobs that have been posted and to retrieve more details about interesting-looking jobs quickly without resubmitting a new request.

I have added line numbers to help explain what the code is doing, but these would not be included in the real version.

EXAMPLE 2 JOBSERVE.XSQL (VERSION 1.0)

```
1. <?xml version="1.0"?>
2. <xsql:query mySearch="wap"
3. max-rows="5"
4. connection="Jobserve"
5. xmlns:xsql="urn:oracle-xsql">

6. <xsql:include-request-params/>

7. SELECT substr(title,1,16) short_title, title, location, skills
8. FROM job
9. WHERE UPPER(title) LIKE UPPER('%{@mySearch}%')
10.ORDER BY first_posted DESC

11. </xsql:query>
```

Line 1 identifies this as an XML document that conforms to version 1.0 of the XML standard.

Line 2 is our first xsql tag which, with the matching tag at line 11, surrounds our XSQL query. It also includes the first attribute of the element – mySearch. This sets a default value for the parameter that may be passed from the user's browser request. i.e. If the user does not supply a value, the string "wap" will be used by default.

Line 3 restricts the number of rows returned by the XSQL servlet to 5. This ensures that the amount of data is limited, bearing in mind the low bandwidth of a WAP phone. This attribute can also be used with the skip-rows attribute to return multiple screens of data on separate trips to the database.

Line 4 specifies the database connection that will be used to submit the query. The connection details are part of the XSQL Servlet configuration, described in the release notes, but it's worth noting that this allows us to specify multiple queries against different databases as part of the same XSQL page.

Line 5 specifies the XML namespaces to be Oracle's, to prevent any potential conflicts with other XML dialects.

Line 6 ensures that any user-supplied value for the mySearch parameter is used in the query.

Lines 7 to 10 should look very familiar as they are an embedded SQL query. The only difference from a standard query is the use of the {@var} syntax to utilise the user-supplied parameter. There is one generated field containing a shortened version of the title, which might be useful for our main page hyperlinks.

We can request this page using Internet Explorer 5 or any other XML-enabled browser and the XSQL Servlet will submit the query to the database on our behalf and will return the 'raw' XML to the browser, as follows.

EXAMPLE 3 UNALTERED XML OUTPUT

```
<?xml version="1.0" ?>
<ROWSET>
      <ROW num="1">
            <SHORT_TITLE>Oracle Database</SHORT_TITLE>
            <TITLE>Oracle Database Administrator –
      InvestmentBanking</TITLE>
            <LOCATION>Luxembourg</LOCATION>
            <SKILLS>Oracle Database Administrator For Financial
            Organisation. PL/SQL, SQL*Plus, Forms 4.5, Reports
            2.5.</SKILLS>
      </ROW>
      <ROW num="2">
            <SHORT_TITLE>Senior Analyst:</SHORT_TITLE>
            <TITLE>Senior Analyst: Oracle & Delphi</TITLE>
            <LOCATION>Leeds, West Yorkshire</LOCATION>
            <SKILLS>Senior Oracle & Delphi …
…
…
      </ROW>
</ROWSET>
```

Looking at the output should highlight one of the advantages of XML – the ability to visually interpret what is going on because the tags are descriptive of the data. But even this isn't readable or useful to the average user of the service, so we need to convert it to our desired format using the XSQL Servlet's built in XSL Transformation (XSLT) processor. We can do this by adding line to our jobserve.xsql page that identify the stylesheets that we want the processor to use to format our data for presentation.

EXAMPLE 4 JOBSERVE.XSQL (VERSION 1.1)

```
1. <?xml version="1.0"?>
2. <?xml-stylesheet type="text/xsl" media="Mozilla/4.0 (compatible; MSIE"
   href="jobserve_html.xsl"?>
3. <?xml-stylesheet type="text/xsl" media="Mozilla/3.0 (compatible;
   HandHTTP 1.1)" href="jobserve_palm.xsl"?>
4. <?xml-stylesheet type="text/xsl" media="UPG1 UP"
   href="jobserve_wml.xsl"?>

5. <xsql:query mySearch="wap"
6. max-rows="5"
7. connection="Jobserve"
8. xmlns:xsql="urn:oracle-xsql">

9. <xsql:include-request-params/>
```

```
10.SELECT substr(title,1,16) short_title, title, location, skills
11.FROM job
12.WHERE UPPER(title) LIKE UPPER('%{@mySearch}%')
13.ORDER BY first_posted DESC

14. </xsql:query>
```

The only lines that have been added to this version are lines 2, 3 and 4 and the only one that is of interest to us from a WAP point of view is line 4. It instructs the XSQL servlet to check the HTTP_USER_AGENT of the request (i.e. which browser is being used to request the page) and, if it contains UPG1 UP, then it will format the data using the jobserve_wml.xsl stylesheet. The XSL Transformation processor will then process our XML data file (example 3) and reformat it according to the instructions we include in the stylesheet file.

So what on earth is the 'UPG1 UP' browser?

One of the potential difficulties with developing for handheld devices is the variety of different device possibilities. Rather than having to go and buy a bunch of mobile phones and PDA devices to test your application, most manufacturers provide free development kits that include documentation and, more important, an emulator of the device itself. This allows you to see how your application will look when running on the device itself. There are a number of emulators that I've used, including the Nokia and Ericcson development kits and, the one used here is the Phone.com UP SDK. The browser used by that device (and a number of others) returns a HTTP_USER_AGENT string containing "UPG UP1". So, if the user is using the phone.com UP Browser, then the data will be formatted using jobserve_wml.xsl, as follows. Again, I've added line numbers for clarity but these would not be in the final version.

EXAMPLE 5 JOBSERVE_WML.XSL STYLESHEET

```
1. <?xml version="1.0"?>
2. <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
3. <xsl:output method="xml" doctype-public="-//WAPFORUM//DTD WML 1.1//EN"
4. media-type="text/vnd.wap.wml"
5. doctype-system="http://www.wapforum.org/DTD/wml_1.1.xml"
6. encoding="ISO-8859-1"/>
7. <xsl:template match="/">
8. <wml>
9. <card>
10.<p><b>RECENT POSTINGS</b></p>

11.<xsl:for-each select="ROWSET/ROW">
12.<p><anchor><go>
13.<xsl:attribute name="href">#card<xsl:number/>
14.</xsl:attribute>
15.</go>
16.<xsl:value-of select="SHORT_TITLE"/></anchor></p>
17.</xsl:for-each>
18.</card>

19.<xsl:for-each select="ROWSET/ROW">
20.<card>
```

```
21.<xsl:attribute name="id">
22.card<xsl:number/>
23.</xsl:attribute>

24.<p><b><xsl:value-of select="TITLE"/></b></p>
25.<p><xsl:value-of select="LOCATION"/></p>
26.<p><xsl:value-of select="SKILLS"/></p>
27.</card>
28.</xsl:for-each>
29.
30.</wml>
31.</xsl:template>
32.</xsl:stylesheet>
```

XSL is a powerful language and can seem a little intimidating at first so it's worth your time to investigate one of the many detailed tutorials at the web sites included in the Resources section, but let's have a brief look at what's going on in this example.

Line 1 identifies this file to be an XML document. So, not only is our data going to be returned using XML, but our stylesheet language is defined as XML and so are WML documents!

Line 2 defines the namespace for XSL.

Lines 3-6 are interesting because they were the only lines that caused any problems when I first started developing the protoype (and sincere thanks to Steve Muench of Oracle for helping me sort this out!). Because WML is an XML dialect in itself, every WML document is an XML document and so we can't just include the following line in the stylesheet, which is the way we normally identify a WML document in the WML file itself.

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
```

Instead, we use the following line to generate the line listed above in the output.

```
<xsl:output method="xml" doctype-public="-//WAPFORUM//DTD WML 1.1//EN"
          media-type="text/vnd.wap.wml"
          doctype-system="http://www.wapforum.org/DTD/wml_1.1.xml"
          encoding="ISO-8859-1"/>
```

Our formatting instructions start with line 7, which instructs the processor that we want to match the root node of the XML document (/), containing all elements of the data.

Then things start to get a little simpler with lines 8, 9 and 10. One way of looking at XSL stylesheets that may help when you first start using them is to think of them as templates of your final document, containing some fixed text that will be passed through to the finished file, along with XSL instructions that will 'insert' your XML data into the template. The <wml> tag in line 8 is the first template item, so that will be included unchanged.

In fact, one of the more straightforward approaches I've found to developing stylesheets is to develop the a mock-up of the end result I'm looking for first, including the data. Then I can see which items are template text (such as the <wml> or <card> tag) and which will be generated using XML/XSL (such as a job title).

Note that this screen will consist of a WML deck with a number of cards, so the <card> tag at line 9 will be the start of the first card, containing a short description of the most recently posted 5 jobs,

which will link to other cards containing further details about the job. Line 10 should look very familiar to HTML, but note that it's essential to have those closing tags!

Now we start to process the XML data. Line 11 says 'for each row in the XML data document', do the following …

Line 12 will output a few WML tags for the row of data and then, because WML is an XML language itself, line 13 will set the *href* attribute of the element we're generating at the moment, the <go> element from the previous line. This indicates the location that the <go> element should jump to if this link is selected by the user.

Because the cards are being generated automatically, depending on the number of jobs returned, I needed to give each job details card a unique identifier. I've used the <xsl:number/> tag to generate unique numbers for each card in the deck so that the links can point to distinct card names in the format card<unique_id> (e.g. card1, card2, …). Note that this data isn't coming out of the database, but is being generated as part of the XSL transformation process.

Line 14 is just the closing tag for the href attribute and line 15 the closing tag for <go> element of the link which identifies the destination.

Line 16 is the first time that we're going to insert any of the data contained in our XML data document, using `<xsl:value-of select=element_name/>`. The short description for the job will be inserted into the anchor element and will be the text that's displayed for the link.

Line 17 ends the 'loop' started at line 11, so all the instructions and template text between these two points will be generated for each row included in the data document and our first WML card in the deck ends at line 18.

Then we make use of another key XSL facility, the ability to reuse the XML information in different ways in different parts of our output document – once for the main menu of hyperlinks with the short title and again for the job details pages. Line 19 is just like line 11, so we're starting another loop through the same data.

The first thing we do is to start a new card for each row in the data document, using xsl:number again to ensure that the card idenitifiers match the links that we generated in our main card. Then we insert various values from the document into the card, close the card off at line 27 and end the loop. So this loop will generate one card for each ROW element in the XML data document.

The resulting WML output follows and can be read by a WAP-enabled phone or emulator. It's worth looking over the examples 3 and 5 together to see how we arrive at the finished article.

EXAMPLE 6 TRANSFORMED XML OUTPUT TO WML DECK

```
<?xml version = '1.0' encoding = 'ISO-8859-1'?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
            "http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
      <card><p><b>Recent Postings</b></p>
            <p><anchor><go href="#card1"/>Oracle Database </anchor></p>
            <p><anchor><go href="#card2"/>Senior Analyst: </anchor></p>
            <p><anchor><go href="#card3"/>Oracle Developer</anchor></p>
            <p><anchor><go href="#card4"/>Oracle Developer</anchor></p>
            <p><anchor><go href="#card5"/>Oracle Developer</anchor></p>
      </card>

      <card id="card1">
```

```
            <p><b>Oracle Database Administrator - Investment
            Banking</b></p>
            <p>Luxembourg</p>
            <p>Oracle Database Administrator For Financial Organisation.
            PL/SQL, SQL*Plus, Forms 4.5, Reports 2.5.</p>
      </card>

      <card id="card2">
            <p><b>Senior Analyst: Oracle &#38; Delphi</b></p>
            <p>Leeds, West Yorkshire</p>
      <p>Senior Oracle &#38; Delphi developers required by large retail
organisation.
      …
      …
      …
</wml>
```

XSL is an incredibly powerful general transformation language and can be used to achieve a wide variety of outcomes using a little imagination. But do you remember these lines that I glossed over in an earlier example?

```
<?xml-stylesheet type="text/xsl" media="Mozilla/4.0 (compatible; MSIE"
href="jobserve_html.xsl"?>
<?xml-stylesheet type="text/xsl" media="Mozilla/3.0 (compatible; HandHTTP
1.1)" href="jobserve_palm.xsl"?>
```

Although I've illustrated WML output here, we could create other stylesheets for other browsers and apply them conditionally, based on the browser used to access the XSQL page. The two lines above instruct the XSL Servlet to use completely different stylesheets based on the user accessing the page from different browsers, the first for Internet Explorer, the second for a PalmOS browser that I used in development. Once the initial work on the XSQL page had been completed, implementing new stylesheets for new devices proved easy and I didn't have to touch any of the underlying data access code (other than to insert a new stylesheet reference in the XSQL page). Here is an example of the stylesheet for Internet Explorer. The output is very basic, but it's useful to compare the stylesheet to generate HTML with the one used earlier to generate WML

EXAMPLE 7 JOBSERVE_HTML.XML

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

        <p><b>Recent Postings</b></p>
      <xsl:for-each select="ROWSET/ROW">
        <p><b><xsl:value-of select="TITLE"/></b></p>
          <p><xsl:value-of select="LOCATION"/></p>
          <p><xsl:value-of select="SKILLS"/></p>
      </xsl:for-each>

    </html>
  </xsl:template>
</xsl:stylesheet>
```

Much simpler, isn't it?

CONCLUSION

Oracle's XML Developer's Kit allows you to develop web applications, including business intranet applications, which use a variety of different presentation formats without constant changes being required to the underlying application logic. It helps you to increase the speed with which new devices and markup languages can be deployed and ensures that you can offer users information in the formats that they require, when they require it.

Earlier in the paper I listed a number of reasons why it is worth developing WAP services. I deliberately avoided one of the major criticisms of WAP, that the applications are awful and why would anyone want access to a service via a tiny little screen, using a keypad to input characters and to look up the football scores? WAP development is at a very early stage, which reminds me of previous technologies that I've witnessed as they develop from laughably under-powered, misunderstood toys until people start to understand their particular strengths and utilise them. If the biggest criticism of WAP is that there aren't any decent applications, then shouldn't we be rushing to develop those applications?

BIBLIOGRAPHY AND RESOURCES

The following Oracle XDK reference document proved invaluable in developing this paper. It can be found at http://technet.oracle.com/tech/xml/xsql_servlet/index.htm

Oracle XSQL Pages and the XSQL Servlet Release Notes for Version 1.0.0.0 (Production)

Steve Muench, Consulting Product Manager and XML Evangelist, Oracle Corporation March 27, 2000

Oracle has a dedicated XML section of Technet at http://technet.oracle.com/tech/xml as well as a user discussion board where enhancement suggestions and technical questions can be posted. (http://technet.oracle.com/support/bboard/discussions.htm)

No subject seems to have wider coverage on the internet at the moment than XML. Here is a small selection of the sites that I've found most useful

W3C XML Specification	http://www.w3.org/XML
IBM XML Developer	http://www.ibm.com/developer/xml
Apache XML	http://xml.apache.org
XML Journal	http://www.sys-con.com/xml/index2.html

There are a large number of online resources dedicated to WAP development. Some of the sites that I've found the most useful are

Nokia WAP on Web	http://www.nokia.com/wap
Phone.Com Developer's Info	http://www.phone.com/developers/index.html
O'Reilly Wireless Developer's Network	http://www.wirelessdevnet.com/
WML Specification	http://www.oasis-open.org/cover/wap-wml.html
WAP Forum	http://www.wapforum.org

A WAP Tutorial http://www-4.ibm.com/software/developer/library/wireless

ABOUT THE AUTHOR

Doug Burns is an independent consultant who has 10 years experience working with Oracle in a range of industries and countries. He has worked as a course instructor for both Oracle UK and Learning Tree International. He can be contacted at doug.burns@lineone.net and this document and related articles will available on his website at http://doug.burns.tripod.com (along with some Sinclair ZX Spectrum games written in a previous life.!)