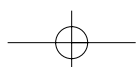
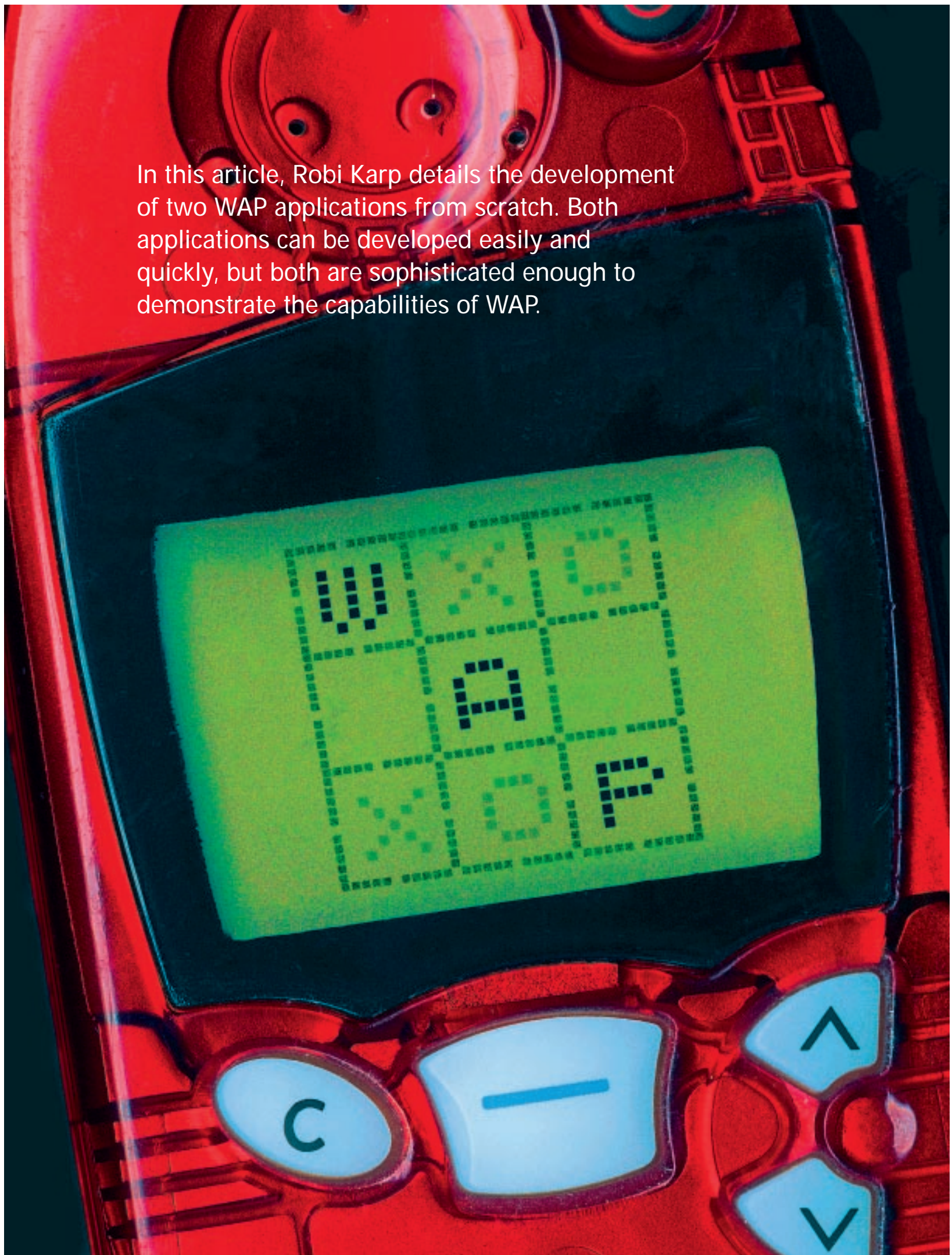
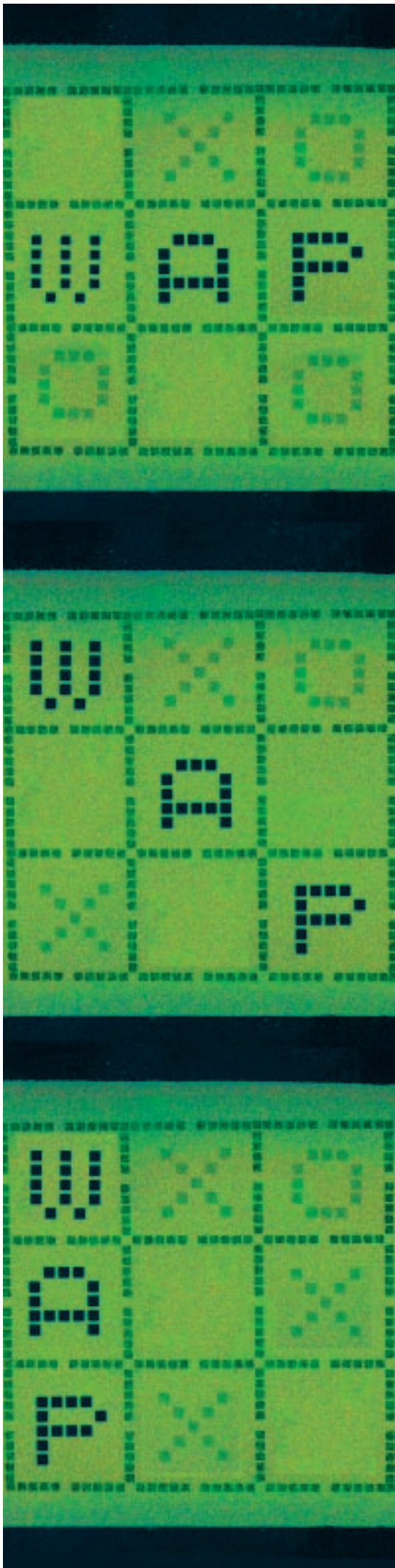
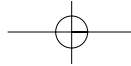


In this article, Robi Karp details the development of two WAP applications from scratch. Both applications can be developed easily and quickly, but both are sophisticated enough to demonstrate the capabilities of WAP.





cover feature

An Insider's View of a WAP Application

WAP, is a standard way for data to be communicated to and from mobile devices. WAP, along with WAP Markup Language (WML) and WMLScript (the scripting language) are intended to be for mobile devices what HTTP, HTML and JavaScript are for the Internet and home computers.

This article steps through the development of two WAP applications from scratch. Both of them are simple enough to develop quickly but are sophisticated enough to be able to demonstrate the capabilities of WAP. This article does not discuss, in detail, the WAP specifications, nor does it present the application source code. The source code is available online from Fluffy Spider Technologies: <http://www.fluffyspider.com.au/wap-article.html>

The first application is a simple tic tac toe game. The game downloads to the browser entirely and no back-end server interaction is necessary.

The second application is a little bit more ambitious. It is a location based look-up service. This application queries the Telstra Yellow Pages Web server for a particular service in a particular area and returns the results to the mobile device. This application

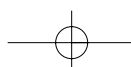
demonstrates true client to server communication.

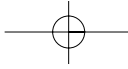
This article was researched during late 1999 and early 2000. Because of this a lot of the tools used are now out of date or have since been upgraded.

In the examples below all of the function and variable names are short and cryptic. While this would normally be considered bad coding style, we use short names as they are downloaded to the browser 'as is' so shorter names consume less bandwidth and phone memory. We also use the variable `$(src)` quite often. This indicates the base URL plus WMLScript file of our software.

Development Environment

Our initial work used the integrated development environment (IDE) from Ericsson. This had some rough edges, for example hyperlinks displayed on lines by themselves and underlines trailed to the end of the line. The main disadvantage to this IDE, however, was that it ran exclusively on Windows, whereas we are exclusively a Linux/Unix development environment. We discovered the Nokia development environment from one of the WAP mailing lists, and it too ran on





Windows but was primarily Java, although we were able to extract the Java out and run it on Linux.

The development environment provided us with WML and WMLScript bytecode compilers (explained further below) along with a text editor etcetera. It also provided a graphical displayer that took the form of a mobile phone (Nokia of course) of your choice. One could select from a 7110, a 6510 and a 6110. It did mention that of those three, only the 7110 could truly support WAP.

Finally, it seems that certain 'standard' functions are not available on all development environments. We tried to use a library function to replace a character at a particular point in a string. We found that it was unavailable on the first environment we used but was available in the Nokia development suite.

WML and WMLScript

WML is a markup language based on HTML and XML. It was designed for mobile devices with restricted network bandwidth, screen real estate and device processing power.

For this reason the language is compact, with as few tags as necessary (less for the client-side browser to be aware of) and the plain text is compiled to bytecode. At this stage the available devices seem to offer varying levels of support for WML. Even the most widely used device, the Nokia 7110, only partially supports WAP 1.1 (WAP 1.1 is the current version, WAP 1.2 is under consideration). In particular, elements such as tables and pictures are not uniformly supported.

As WAP must run on many different types of displays, from those with two lines to larger screens, the specifications are general with respect to layout. They state that the layout is primarily up to the device.

Given the above two points, WAP designers must be careful to take into account the terminal device when designing and implementing WAP applications. A software interface to WML that takes this into account for you is a better solution to this portability problem. FST has been working on such a product.

For us the bytecode compiler became an issue as we didn't have one outside of the IDE. We needed one to generate WML dynamically (for the same reason that CGI programs generate HTML) so we ended up reverse engineering some generated bytecode to determine the encoding. It might have been easier to look at the specifications (<http://www.w3.org/TR/wbxml>) but at the time this was the quickest way for us.

A deck is the smallest WML element and is the equivalent to an HTML page. Decks are made up of cards which are

logical user interaction elements. One downloads the entire deck (many associated cards) at once, allowing traversal through the cards easily without having to query the server.

Each card has a unique identifier to allow navigation from another card or deck as well as a title, displayed on the top of the screen, and optional attributes including newcontext which informs the browser to reinitialise itself and clear the browser variables.

```
<card newcontext="true" id="splash" title="Tic Tac Toe">
```

Each card may have events defined. The most common of these are the accept or forward event and the previous or backward event. These can be defined at the deck (global) level and overridden at the card level. This method was used in the tic tac toe game, we had a global previous event defined to go to the 'do you want to quit?' card where we overrode it to do nothing. More than one action may be defined for an event, and in this case the user is presented with a menu of options. We avoided doing this entirely as it does not make for a good user interface.

Size of compiled code is an issue for the developer, different devices or browsers have different limitations.

We ran some tests on the sizes of different code blocks.

Test 1: This version of separate <p> blocks with rest of code came to 18 Bytes.

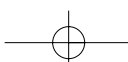
```
<p align="center"><b>- Board User-</b></p>
<p align="center">$(line1)</p>
<p align="center">$(line2)</p>
<p align="center">$(line3)</p>
```

Test 2: This version came to 9 Byte.

```
<p align="center"><b>- Board User-</b>
<br/>
$(line1)
<br/>
$(line2)
<br/>
$(line3)
</p>
```

Table 1: Microbrowser Specs

Browser	Compiled WML deck limit
UP.Browser 3.2	1,492 Bytes
UP.Browser 4.x	2,048 Bytes
Ericsson R320	Approx 3,000 Bytes
Ericsson R380	Approx 3,500 Bytes
Ericsson MC218	More than 8,000 Bytes
Nokia 7110	Approx 1,500 Bytes





The figures only represent the formatting code, not the identifiers or string. The reason the second test came out smaller is that the Nokia compiler puts a single space before and after any text on a single line that is not touching any other directive tags.

We also noticed that if the line variables are not assigned in Test 1 then nothing is displayed whereas Test 2 displays three blank lines. This could be important in some situations.

WMLScript

WMLScript is an extremely powerful scripting language that is to WAP what JavaScript is to HTTP. Strings, floats and 32-bit integers are supported as well as an 'invalid' type that represents nonsensical results. On devices without a floating point processor some functions are not available, but these can be tested for at run-time. There is also full boolean evaluation and a mathematical library as well as some string manipulation functions available.

Unfortunately the language only has string arrays but these may be used to store other types. String tokenising functions break the string up at specified delimiters, and this functionality could be used to store any type. It would not be very efficient though.

Standard libraries allow interaction with the browser and the hand-set as well as the math and string functions mentioned above. For example, there is a WMLBrowser.go() function. This instructs the browser to go to a card: WMLBrowser.go("#aCard");

This function only goes to the specified card after the rest of that function has been executed, it is not like a function call that returns when finished.

The language has the extern keyword which allows a function to be called from outside the file in which it was written. Variables may be declared anywhere within a function and stay in scope until the function exits, but they may not be declared outside of a function.

Some simple examples:

```
var beg = String.subString(board, 0, loc);
var end = String.subString(board, loc + 1, 27);
```

These examples extract substrings from the variable 'board'. The first one extracts 'loc' characters from the first (zeroth) element and assigns that to 'beg'. The second extracts the characters from 'loc' + 1 to the 27th character from the 'board' variable and assigns them to 'end'.

Below is an example of a local function (there is no extern keyword) that extracts portions of a string passed as a

parameter (note no var used in the function header) and sets browser variables with them.

```
function setBoard(pieces)
{
  //
  // Extract the 1st, 2nd and 3rd 9 element
  // substrings from the given parameter.
  // Set the browser variables line1, line2
  // and line3 to the results.
  //

  var line;

  line = String.subString(pieces, 0, 9);
  WMLBrowser.setVar("line1", line);

  line = String.subString(pieces, 9, 9);
  WMLBrowser.setVar("line2", line);

  line = String.subString(pieces, 18, 9);
  WMLBrowser.setVar("line3", line);
}
```

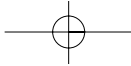
Tic Tac Toe

This application is a simple tic tac toe game, it downloads all of the WML and WMLScript to the browser at once and runs entirely on the browser, no access to the server is necessary. The code for this game is divided up into two source files, a WML file and a WMLScript file.

The biggest problem with this game initially was simply getting input from the user. The actual input, not the navigation, requires several key strokes. A simplified text entering example is an input box. This is displayed as two square brackets (in our development environment) and any default string will be placed there initially. To enter text into the box or edit it, you move the cursor over the text or box (it becomes highlighted) then press the 'edit' button to edit the text – if there is more than one action defined then you get a list of options. On the Nokia simulators, the screen is now replaced with an edit/enter screen. This is the same screen you see when you want to enter any other text on a Nokia 'Human Technology' hand-set.

We spent several days trying alternative methods to make text entering simpler and faster. We would have liked to simply enter a number corresponding to the player's desired position. The number five for centre, one for top left, nine for bottom right etcetera. We did end up using this numbering scheme to position mapping but only via the input text screen described above.

An alternative we might have used was to have nine hyperlinks across the board. This would have allowed the player to move the cursor to the desired location and press the 'accept' button. As the game progressed the links could be



removed to speed things up. At the time we couldn't think of any way of removing the links without going back to the server. However, we now know that if a link has no text it is not visible and the cursor will not stop at it.

```
<a href="yp.wmlsc#func()">$(linkvar) </a>$(selected)
```

When the position has not been selected, the linkvar variable is set to the text of the link, the selected variable is set to an empty string and this text is shown. When the position has been selected, the linkvar variable is set to an empty string, the selected variable is set to either a nought or a cross which is now shown.

The disadvantage of using this method is that the player has to press the down button nine times to get to the bottom right hand corner. We considered this to be unacceptable.

We used a similar approach to displaying the board itself. We maintained three variables, one for each line of the board, and when selections were made, the text of the line was updated by putting a nought or a cross in place of a star. The Nokia 7110 simulator used proportional fonts to display the page which meant that with some combinations lines didn't match up.

We now know that it would have been easier and would work better to set the board up with nine variables that will display one of three pictures or letters (nought, cross or star, for not yet selected). The trick is to ensure that the server does not need to be contacted during a game.

Mobile Yellow Pages Query

The mobile application queries the Telstra Yellow Pages Web server for the details of some criteria, such as hardware store or bank, in an area specified by the user. It has a true client server architecture. The client is the WAP application running on the terminal device within a browser. The server is a CGI style program running on a WAP/Web server. The user selects one of a finite set of criteria, enters the suburb or post-code and sends the query to the server. The server converts the suburb to a postcode (if a suburb was entered) forms a query URL and uses the command line program wget, available on most Linux systems, to contact the Telstra Yellow Pages Web server. The server program reformats the response from the Telstra Yellow Pages Web server and sends it out to the client WAP application as a deck of information.

Once again, the WML was byte compiled as in the tic tac toe game, by using the reverse engineered IDE compiled byte code. We wrote functions to generate the byte code as

needed. For example the following function creates a NULL terminates inline string:

```
void
WMLinLineStr(const char* str, int* total)
{
    //
    // Output an inline string in WML bytecode format.
    // Character 3 followed by a NULL terminated string.
    //
    printf("%c%s%c", 3, str, 0);
    *total += 2 + strlen(str);
}
```

Special characters, such as ampersand, do not need to be escaped for this sort of conversion.

This application was almost exclusively one of string manipulation. Despite this, we did not take advantage of a string table that compiled WML offers to cache duplicate strings. First we were hand compiling the code, although this is not prohibitive, but secondly, we did not feel it was necessary. It would have been unlikely to have the same string more than once.

We had to ensure that we limited client bound data to 1.4kb. Whereas this is not a limit specified by WAP itself, it is a limit of the deck size of Nokia hand-sets, our testing environment, and a limit that most developers on the mailing lists are working to. This is the sort of portability issue that WAP developers must be aware of. Once again, a device API would be of great benefit, to overcome this limitation we only sent results until the size of the deck was too big, a link was then added to the next set of results

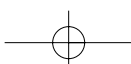
Maintaining state in CGI (client/server) WAP applications presents the usual stateless connection problems but without the benefit of cookies. The way to maintain state within a session is to set variables to the desired values (some sort of authentication key/value pair) and pass them back to the server for verification with any other data that is sent either as a variable/value pair or as part of a constructed URL. For example, the link to the next page of results for our application contained a generated URL – this is not a new concept to the CGI programmer.

Coding Bits and Pieces

Dynamic Hyperlink Modification

We refer, above, to a method of seemingly being able to change the location of a hyperlink without having to regenerate the page from the server. This is an example of a card that uses this method. We used this in the Yellow Pages Query application.

```
<card id="where" title="Your Location">
  <do type="prev">
```





```

        <prev/>
</do>

<p>
  Enter postcode / suburb of area to search in:
  <input type="text" title="Area" name="loc"
    emptyok="false" format="M*" />
</p>

<p align="center">
  Surrounding areas:

  <!-- If any of the variables used below are set then
  they are displayed. If, however, only one of them is set,
  then only it is displayed. If that is the variable $(yn)
  then the hyperlink is set. If not, then only static text
  is displayed.

  See notes, below.
  -->

  <b>$(bf)&nbsp;</b>
  <a href="$(src)#ar()">$(yn) </a>
  <b>&nbsp;</b>$(af)</b>
</p>

<p align="center">
  <a href="$(src)#get()">Do Search</a>
</p>
</card>

```

In the above card example, we ask users if they want to search surrounding areas. The result of the question is stored in the $\$(yn)$ variable and is initialised to 'yes'. The screen will display the following:

```
Surrounding areas: [Yes] No
```

The square brackets indicate that 'yes' is the currently selected value. 'No' is underlined to indicate that it is a hyperlink. The only option the user can now select is the hyperlink to a function that toggles the value of the $\$(yn)$ variable. This would cause the display to be:

```
Surrounding areas: Yes [No]
```

The 'yes' is now the hyperlink (underlined) and 'no' is selected.

We use three variables to be able to change the position of the hyperlink while maintaining the order of the 'yes' and the 'no' without needing to regenerate the page. In our code the three variables are $\$(bf)$, $\$(yn)$ and $\$(af)$. The variable $\$(yn)$ is the hyperlink. In the first case $\$(bf)$ has the value 'Yes' and $\$(yn)$ has the value 'No' and $\$(af)$ is set to an empty string, and hence not displayed. In the second case, $\$(bf)$ is empty, $\$(yn)$ has the value 'yes' and $\$(af)$ has the value 'no'.

The function that alternates the yes and no text follows:

```

extern function ar()
{
  //
  // Toggle variables to search surrounding areas
  // and update the display.

```

```

//
var yn = WMLBrowser.getVar("yn");

if (yn != "No")
{
  WMLBrowser.setVar("bf", "[Yes]");
  WMLBrowser.setVar("yn", "No");
  WMLBrowser.setVar("af", "");
}
else
{
  WMLBrowser.setVar("bf", "");
  WMLBrowser.setVar("yn", "Yes");
  WMLBrowser.setVar("af", "[No]");
}

WMLBrowser.refresh();
}

```

Saving Keystrokes and Bandwidth

In our Yellow Pages Query application we set a variable to indicate the location the user is currently searching in. This allows users to make multiple searches without having to enter their location each time, they only have to set it the first time and whenever they wish to change it. We can construct a target URL from within a script as follows:

```

WMLBrowser.go("#yp.cgi?loc=" + loc + "&yn=" + yn + "&opt="
+ opt);

```

By constructing the URL as opposed to using a form-style approach we can check the validity of the current location (is it set or not) rather than sending an empty location back to the server and returning from there with an error message.

Timers

One is able to set timers in WML. Each card may have one timer. The `ontimer` attribute of the card tag specifies that after the specified time the given URL is to be loaded. Timers are in tenths of seconds. In this example, the `init()` function is loaded after three seconds (30 tenths). We allow the user to not wait by pressing the 'enter' button (which is bound to the 'accept' event). This jumps them directly to the `init()` function. The 'prev' event has a `<noop/>` tag specified, preventing the user from backing out of this card.

```

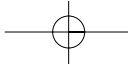
<card ontimer="$(src)#init()" newcontext="true"
  title="Loading...">
  <timer value="30"/>

  <do type="accept" label="Enter">
    <go href="$(src)#init()"/>
  </do>

  <do type="prev">
    <noop/>
  </do>

  <p align="center">

```



```


<br/>Fluffy Spider
<br/>Technologies
</p>
</card>

```

The Form

The following card shows how to use HTML style forms.

```

<card id="card1" title="Title">

  <p>
    // Stuff before...

    <anchor title="Submit">Submit
      <go href="thing.cgi" method="post">
        <postfield name="name1"
value="$var1"/>
        <postfield name="name2"
value="constVal"/>
      </go>
    </anchor>

    // Stuff after...
    // page text...

  </p>
</card>

```

In this example the CGI thing.cgi is called with the two name/value pairs. The first is name1 which has a variable value given by the variable \$var1. The second is name2 which has a fixed value, 'constVal'.

WAP applications are not hard to write. It is the ideas behind those applications which are the challenge to create. As voice and data converge to small hand-held devices I have no doubt that standards and specifications will also change to suite. We are only at the very start of an exciting road of technological innovation.

The following references can be read for more information: *Official Wireless Application Protocol*, Wiley; <http://www.wapforum.com>; <http://www.4k-associates.com/IEEE-L7-WAP-BIG.html>; <http://wap.colorline.no/wap-faq/>; <http://www.ericsson.se/developer-szone/index.asp>; <http://www.waptastic.com/>; <http://www.openwap.org/>; and http://www.gsmworld.com/technology/wap_06.html. ■

Robi Karp is director of Fluffy Spider Technologies (<http://www.fluffyspider.com.au>) a software development company specialising in the field of mobile telecommunications. I have to thank Nick for all his help in putting this article together.

