Under Construction: Wireless Application Protocol

by Bob Swart

Delphi is great for writing internet and intranet applications. In fact, in my view, the WebBroker and InternetExpress way of writing web server applications is second to none. It's also easy to extend or enhance them, for example in order to generate WML (Wireless Markup Language) for WAP (Wireless Application Protocol) applications, as we shall see in this article.

WAP

WAP is used by WAP-enabled mobile phones. It allows users of these mobile phones to access information from the web, provided it has been coded using WML (which is very similar to HTML, as we'll discover in a moment). This means that not everything on the web is available to WAP phone users, but only those special parts that are written in WML. To be honest, you don't want to see the average website on a 200x200 pixel display capable of a mere four shades of grey anyway! WAP is an evolving standard, currently at version 1.2. The Blueprint prototype that we use in this article is WAP 1.2 compatible, as you'll find out in a moment.

Nokia WAP Toolkit

Before we start you need to have some environment to run your WAP applications. Nokia has developed a special WAP Toolkit, currently at version 2.0, written in Java. It requires the Java Runtime Environment 1.2.2 or higher and can be downloaded from www.forums.nokia.com after you've registered yourself on this website. The WAP Toolkit can be used to load static or dynamic WML files (generated by Delphi, for example) and display them in a WAP simulator (see Figures 2 and 3). The WAP Toolkit is actually a very advanced environment and supports far more than we will use in this article, so it's a good way to get started and get more involved with WAP applications anyway.

WML

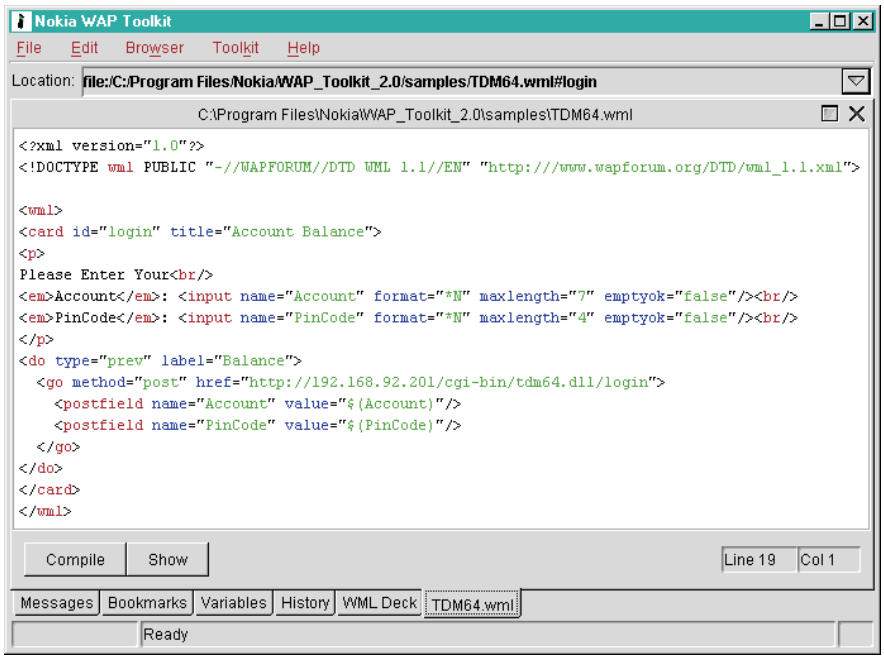
So, what's this WML all about? Or rather, what is not in WML that is in HTML? Well, fancy things like nested tables, layers and so on are out, and so are colours (most WAP phones are capable of showing just four shades of grey). As we look closer, we'll see that WML is actually based on XML (eXtensible Markup Language). WML files aren't called pages (like HTML pages), but rather cards, or a deck (that is, a collection of cards). A card specifies a single interaction with the end-user, while a deck contains an entire session, or multiple sessions, each made up of multiple cards.

An example WML file to login to your bank account with an

➤ *Listing 1: Account Balance WML example.*

```
<?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="login" title="Account Balance">
<p>
Please Enter Your<br/>
<em>Account</em>: <input name="Account" format="*N" maxlength="7"
  emptyok="false"/><br/>
<em>PinCode</em>: <input name="PinCode" format="*N" maxlength="4"
  emptyok="false"/><br/>
</p>
<do type="prev" label="Balance">
  <go method="post" href="http://192.168.92.201/cgi-bin/tdm64.dll/login">
    <postfield name="Account" value="$(Account)"/>
    <postfield name="PinCode" value="$(PinCode)"/>
  </go>
</do>
</card>
</wml>
```

➤ *Figure 1: Nokia WAP Toolkit with my sample card.*



account number and pin code, in order to obtain a real time account balance update, is shown in Listing 1.

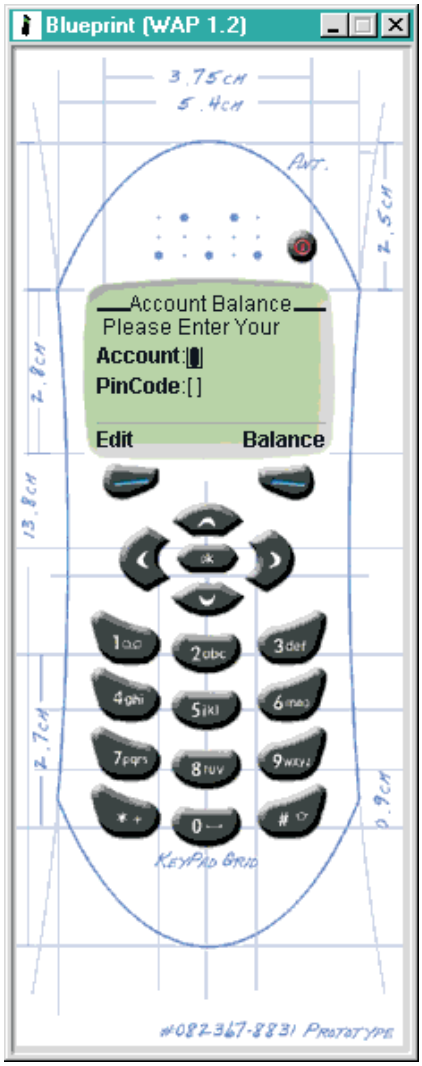
Note the similarities and differences with HTML. Tags must be lower case (which is a rule from XML) and everything on the first page of the screen must be included within `<p>..</p>` tags: I don't know why, but it seems to be a rule as well. Note that each WML document can contain one or more cards, one after each other in the same file, which is then also called a deck.

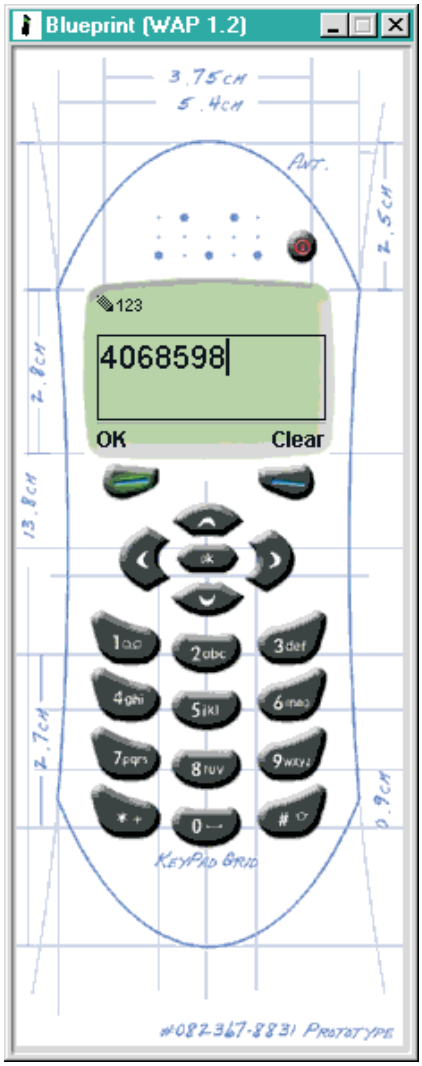
Note that the input types are slightly different, and allow us to specify the format: `*` means an unlimited number of characters of a certain type, where `N` is numerical, `X` is an uppercase letter, `x` is a lowercase letter and `A` is `X` plus punctuation. This is useful to enforce the input of numerals only, as in the account number and pin code in this case, especially in combination with the `maxlength` attribute.

By the way, there is a special input type called `password` which replaces every key you input on your WAP phone with an asterisk. This might seem helpful, but I've learned that there are 'slow' WAP phones which display the entered character for a split second before replacing it with the asterisk. Besides, it's hard to type in your password using the numerical keys (where pressing the 7 key two times gives a Q... or was it three times?), especially if you can't see the current character on your phone's display, so I would recommend careful consideration before you start 'protecting' your WAP applications too much. Most people should be well enough aware of security to not let the entire world look over their shoulders when they enter a pin code on their WAP phone display in the first place.

More interesting are the `<do>` and `<go>` tags. The `<do>` tag specifies a certain action and label that will be associated with one of the two 'soft' buttons on the WAP phone. We can assign a label and action to the left button (next) or to the right button (previous). By default, when you use input fields (such as the account and pin code fields in our example), the left button (next) gets the caption `Edit` and is used to enter or edit the value of these input fields. So, in this case, the right button (previous) is used to assign the `Balance` action to, which is what I have done in the `<do>` tag.

The `<go>` tag is used to jump to an external URL (or WML file) or, in this case, to a web server application: the tdm64.dll ISAPI DLL which generates dynamic WML. This is similar to the `<FORM>` HTML statement, the difference being that we need to explicitly list the fields and values that we want to send along with the call to this server application. In WML we do this using the `<postfield>` tag. Note that we can use existing input fields, in this case the `Account` and `PinCode` fields, using the `$(...)` syntax.





➤ *Figures 2 (left) and 3 (above): The Nokia Blueprint Phone (WAP 1.2).*

WMLScript

Where HTML has JavaScript for client-side scripting support, WML is equipped with WMLScript. Both are, in fact, based on ECMAScript, so you'll find them quite similar. WMLScript can be used for validation of user input, which can be quite important in the mobile world, since you don't want to do all your validations on the server side (think of the bandwidth use and performance). This time, however, I won't be covering any WMLScript, but focusing on how we can use Delphi and WebBroker to generate dynamic WML in response to a request by a card such as the one in Figure 1.

Account Balance

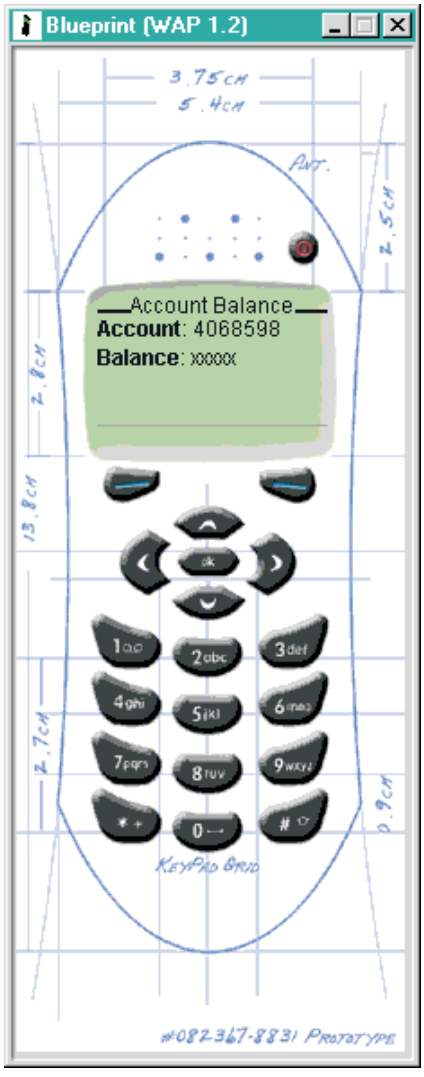
Let's walk through our account balance example, using the WML

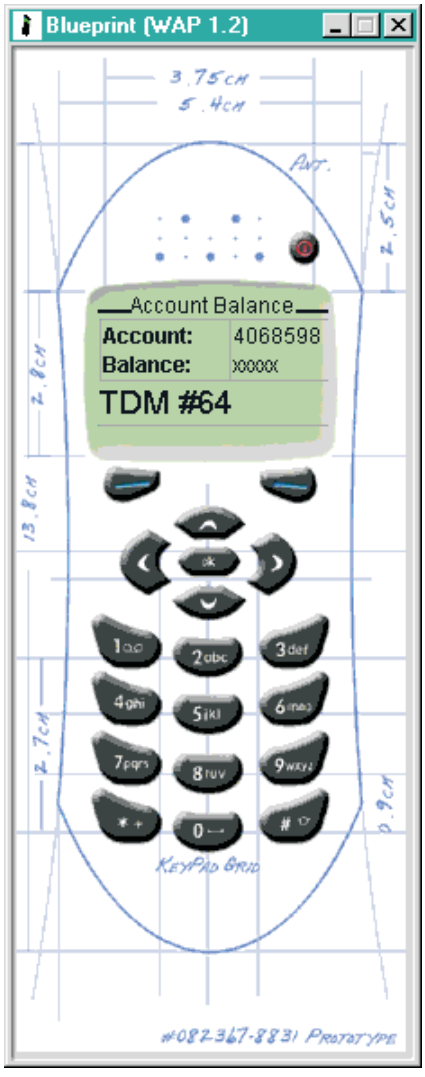
code from Listing 1. As soon as this WML card is loaded, the WAP phone shows the account balance welcome screen, which allows the client to login using the account number and pincode.

Note that the next and previous buttons have a special meaning now. Next is now `Edit` (to insert or edit the account number or pincode), while previous is now `Balance`, or the button to actually (attempt to) login and show the requested balance.

If we load the WML card from Listing 1 into the Nokia WAP Toolkit and click on the `Show` button, the Blueprint phone will display the account balance screen from Figure 2. When you click on the `Edit` button, you can enter the account number (see Figure 3 for my giro account number: donations welcome!). As soon as you click on the `Balance` button, the web server application tdm64.dll is called, which is next on our schedule to create.

➤ *Figures 4 (below) and 5 (right): Account Balance dynamic WML output.*





Dynamic WML

Start Delphi 5 and create a new WebBroker project using the `File | New Web Server Application` wizard. For a WAP application, I would always use an ISAPI DLL, so at least mobile phone users don't have to wait for a CGI application to load and unload before they see the result on their display. As you can see in the WML listing already, the name of the WebBroker application should be `TDM64`, so the ISAPI DLL is named tdm64.dll. We also need a `WebActionItem` with `PathInfo /login` to obtain the information (account and pin code) and return, obviously, the current balance information.

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.ContentType := 'text/vnd.wap.wml';
  Response.Content := '<?xml version="1.0"?>'#13#10 +
    '<!DOCTYPE wml PUBLIC ' +
    '"-//WAPFORUM//DTD WML 1.1//EN" '+
    '"http:///www.wapforum.org/DTD/wml_1.1.xml">'#13#10#13#10 +
    '<wml>'#13#10 +
    '<card id="Balance" title="Account Balance">'#13#10 +
    '<p>'#13#10 +
    '<em>Account</em>: ' +
     Request.ContentFields.Values['Account'] + ' <br/>' +
    '<em>Balance</em>: xxxxx'#13#10 +
    '</p>'#13#10 +
    '</card>'#13#10 +
    '</wml>';
end;
```

➤ *Listing 2: OnAction generating WML.*

Inside the `OnAction` event for the `WebItemAction`, we need to generate the dynamic WML. But first we need to specify that it is in fact WML that we want to return and not plain HTML. This can be done by assigning `text/vnd.wap.wml` to the `Response.ContentType`. And we also want to include the `DOCTYPE` line, after which an empty line separates the header from the real WML content, which should be encoded as a card again. The example WML output that I want to generate is as follows:

```
<wml>
<card id="Balance"
  title="Account Balance">
<p>
<em>Account</em>: 4068598
<em>Balance</em>: xxxxx
</p>
</card>
</wml>
```

Apart from the fact that we need to perform account number and PinCode validation, the `OnAction` item is implemented as shown in Listing 2.

The dynamically generated WML produces the expected result on the WAP display, as can

be seen from the screenshot in Figure 4.

WebBroker And WML

So far, we've seen what WAP is (a wireless application protocol), how we can write WML, and how we can generate simple WML, but it would be interesting to see if we can somehow use the existing WebBroker components to produce WML. We've already used the `Request` object (remember the `ContentFields` that were posted with the `postfield` tag) and the `Response` object as if we didn't know any better. In fact, these objects will work the same whether we're producing HTML, WML or whatever format we like, and whether we're connecting to a WAP phone or using HTTP. Next time, we'll go deeper into that last topic and cover WAP Gateways, which are needed to run WAP applications in the outside world (rather than the Nokia WAP Toolkit simulator).

Another topic I want to leave for next time is the question about images (who wants to use a WAP phone to display an image anyway?). While a WAP phone is no browser, and has no internal support for GIF or JPEG files, for example, it is still perfectly possible to show bitmaps on the display, using a format called WBMP.

WebBroker Components

For `PageProducers` and (hence) the `DataSetPageProducer` this is no problem. The `MidasPageProducer` will not work, for several reasons, like the fact that the current WAP phones can probably hold up to 10Kb (or perhaps a little bit more) in their memory, and the fact that the generated HTML/XML is dependent on JavaScript files, etc. These are enough reasons not to try to connect a `MidasPageProducer` to a WAP phone, but other `PageProducers` work just fine.

When it comes to the `DataSetTableProducer`, the big question is if the HTML `<TABLE>` tag is compatible with WML. The answer is yes, but with some limits. Obviously, WAP phones have much smaller screens. Also, a number of table attributes have no meaning anymore, and some *must* be used, like the columns attribute (so it's easier for the WAP phone to start rendering the table).

```
<wml>
<card id="Balance" title="Account Balance">
<p>
<table columns="2"><tr>
<td><em>Account:<br/>Balance:</em></td>
<td>4068598<br/>xxxxx</td>
</tr></table>
<big>TDM #64</big>
</p>
</card>
</wml>
```

➤ *Listing 3*

By the way, when the output is too wide or long to fit on the display, you can always use the scroll buttons (the four arrows on the Blueprint phone) to navigate. Figure 5 illustrates the display of the Blueprint phone when showing the account and balance information in a simple HTML/ WML table. The syntax used to produce the display of Figure 5 is shown in Listing 3.

Note the `<big>` tag used here for the first time, which results in a much larger font (and generally takes up too much space to allow its use for more than about a dozen letters).

Unfortunately, all this means that the HTML code that the `DataSetTableProducer` and `QueryTableProducer` are producing is not likely to be WML-compatible for use with WAP phones. So, apart from the simple `PageProducers`, there's a whole area of new WAP/WML-enabled components that can help us to produce dynamic output, however small in amount, much more easily. Especially since the use of tables and queries is not something we must underestimate for WAP-enabled WML-producing applications.

Next Time

Last time we saw that producing XML is in fact quite similar to producing HTML. And this time we've seen that producing WML is just as easy. Next time we'll continue our coverage of WML by showing more ways of producing WML with Delphi (for example, from an ASP application), and by writing some WML producing components to help develop WML applications the Delphi way. Finally, I'll tell you all about WAP gateways, and hope to be able to tell you how to connect to a real WAP application (for those lucky few that already own a WAP phone, that is). All this and more next month, *so stay tuned...*

Acknowledgements

I want to thank my TAS-AT DOC colleague Arnim Mulder for his welcome assistance and suggestions while playing with WAP and WML.

Bob Swart (aka Dr.Bob, www.drbob42.com) is an @-consultant for TAS Advanced Technologies and co-founder of the Delphi OplossingsCentrum (www.tas-at.com/doc), as well as a freelance technical author and speaker at Delphi events all over the world.