Under Construction: Wireless Application Protocol, Part 2

by Bob Swart

Last month I introduced WAP (Wireless Application Protocol) and WML (the Wireless Markup Language) to you, and we saw how we could write dynamic WAP applications using Delphi. We also noticed a number of limitations of WML, and discovered that not all of the WebBroker components produce WML-compatible output.

This time we will cover WML images (in the WBMP format), showing how we can produce dynamic images on your WAP phone. We'll also be building a WML-compatible `DataSetTableProducer` WebBroker component.

WAP Gateway

First, however, I want to start with some important information about deploying WAP applications and WML pages that wasn't covered last time.

Those of you who already tried the examples from last month may have experienced some difficulties with your web server. This is caused by the fact that the special mime types for WML (and WBMP) are not known by normal web servers. You first need to register these mime types at your web server, otherwise WML files and WBMP images will not be handled correctly (and this is also the reason why not all web servers are able to correctly serve WAP applications).

Using Microsoft Internet Information Server (IIS) version 4, you need to go to the Internet Service Manager dialog, and click on the `Properties` button for the (Default) Web Site. In this dialog, you need to go to the `HTTP Headers` tab and click on the `File Types` button in the lower right corner. In the new pop-up dialog, you can enter additional mime types that your web server should support. For WAP applications, you need to enter two new mime types: first, the associated extension .wml, which should point to content type `text/vnd.wap.wml`, and second the associated extension .wbmp which should point to content type `image/vnd.wap.wbmp`. Your web server will now function as a WAP application server.

The mirror of my website (at www.drbob42.co.uk) is hosted by TDMWeb, which has the above mime types registered, so I can use this mirror website as a test environment for my WAP applications. I have uploaded some examples to this website.

WAP Images

As a first example, I want to show you how to return (or produce) images inside WAP applications. In normal web applications, these can be of type GIF, JPG (or JPEG) or PNG, but WAP applications can only use WBMP files. WBMP stands for Wireless Bitmap and is a new image format. WBMP files which are currently supported (type 0) are two colour (black and white) and use no compression. More colours can be added later, when WAP phones get more capabilities, and compression is also an option that might be possible for future types of WBMP. But remember that the current generation of WAP phones has little processing power to decompress the image, so it may take a little while for a new WBMP format to emerge.

Because only the latest graphical design tools will support WBMP, the easiest way I found to use WBMP files is to first generate a two-colour file in GIF, JPEG or BMP format using something like PaintShop Pro (or Resource Workshop), and then use a converter to generate a WBMP file. If you don't have or know how to find such a converter, you can use a free online version on the web at www.teraflops.com/wbmp which can load a GIF, JPG or BMP file from your disk and convert it into WBMP, ready to be used inside a WAP application.

The WML that you must use for WBMP files has the following syntax (for a static image):

```
<img src="http://domain.com/
  logo.wbmp" alt="logo"/>
```

Alternatively, and more interestingly, there's a way to produce dynamic WBMP. First, you need a slightly modified WML statement:

```
<img src=
  "http://domain.com/cgi-bin/
  tdm65.exe/image" alt="logo"/>
```

And then you need to add a `WebActionItem` with `PathInfo` set to `/image`, so it can be called to produce a dynamic WBMP image. This is done in the Listing 1 code snippet (note that the web server must be able to return files of the specified mime type, as I've outlined above).

➤ *Listing 1: Returning dynamic WBMP image.*

```
procedure TWebModule1.WebModule1WebActionItem3Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  ImageStream: TFileStream;
begin
  ImageStream := TFileStream.Create('d:\www\logo.wbmp', fmOpenRead);
  ImageStream.Position := 0; // reset ImageStream
  Response.ContentType := 'image/vnd.wap.wbmp';
  Response.ContentStream := ImageStream;
  Response.SendResponse;
end;
```

Note that we don't need to free the `ImageStream` in the above listing, since the `Response.ContentStream` will now own the actual file and will free it when the `Response` object is destroyed. The result of producing the dynamic WBMP image can be seen in Figure 1.

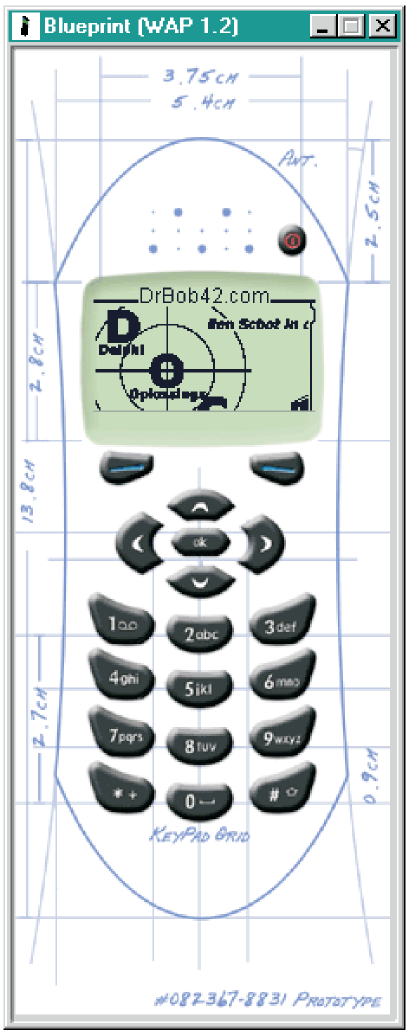
In practice you may want to grab this image from a database, or some other location, but at least this example has shown how you can produce the 'second' WAP-related mime type using WebBroker.

TWMLDataSetTableProducer

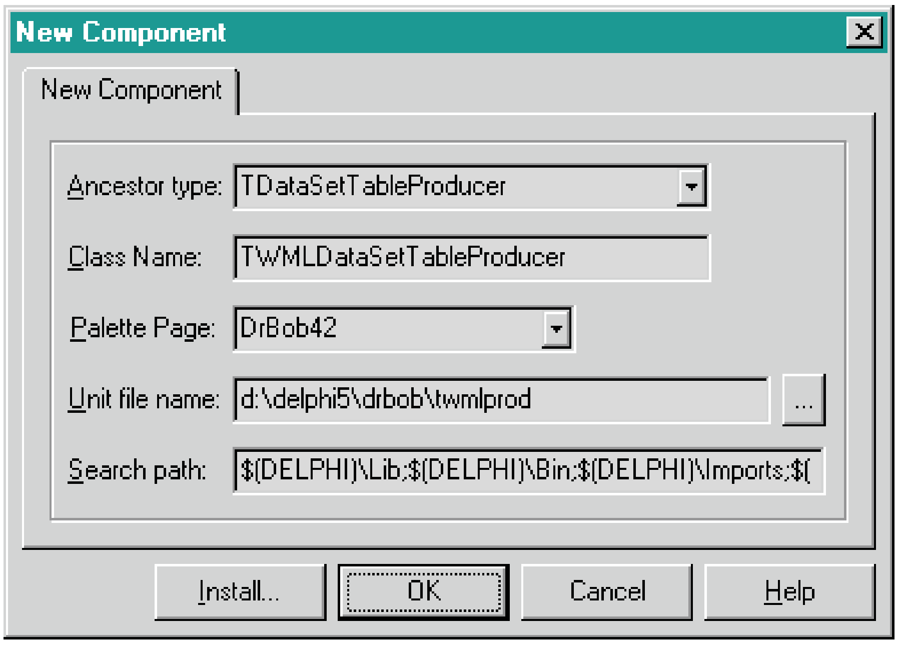
Last time, we noticed that the `PageProducers` were compatible with WML and could be used in WAP applications. We also saw, however, that the `TableProducers` produced HTML that was not compatible with WML. The differences were twofold: some HTML tags were produced in upper case (while WML demands lower case), and some attributes were used (like `bgcolor` and `border`) that were unknown by WML. As a third difference, I noted that the `columns` property was not used in the `DataSetTableProducer`, but is a required attribute in WML.

In short, there are several things we need to override in the original version of `TDataSetTableProducer`, so start with `File | New` and select `New Component` to start the dialog for which you need to specify the `TDataSetTableProducer` as parent (ancestor type) and the class name `TWMLDataSetTableProducer`. I have used the `DrBob42` palette page, but you may prefer to put it on the `Internet` tab instead.

The original `TDataSetTableProducer` component can be found in the dbweb.pas unit (for those of you who have the VCL source code). It consists of a number of methods that produce uppercase HTML tags, such as `Content`, `TableCaption` and `TableHeader`. The `Content` method is the one that actually produces the WML (previously HTML) output. It calls an important function to do the actual work, called `HTMLTable` inside dbweb.pas. In our case, I've rewritten this function and called it `WMLTable` (to make absolutely clear that it generates WML instead of HTML). The new `WMLTable` function calls supporting methods from the new `TWMLDataSetTableProducer`, such as `TableHeader` and `FormatCell`. Originally, the `HTMLTable` function also called `TableCaption`, but since that has no use in WML, I've decided not to call that method, so we don't even have to override it in the new `TWMLDataSetTableProducer` class, leaving only `Content` and `TableHeader`. Like I said a few lines ago, the `Content` requires only a minor change, namely a call to `WMLTable` instead of `HTMLTable`. The `TableHeader` method is completely rewritten (see Listing 2), since it must return a lowercase `<table>` tag, and should only include the 'columns' attribute (and not produce the other attributes that used to be generated by the `TableHeader` method of the parent class). A third method called `FormatCell` has been designed with tag case-sensitivity in mind, since the actual tag (`TD` in the old case, `td` in our case) is passed as argument, so we only need to change the call to `FormatCell` from within the `WMLTable` method to ensure WML-compatible output is generated. The final method, called `TableCaption`, is something that is not used in WML, so instead of overriding it (and returning nothing), we can just make sure not to call it.



➤ *Figure 1: WAP with WBMP image.*



Ancestor type: TDataSetTableProducer
Class Name: TWMLDataSetTableProducer
Palette Page: DrBob42
Unit file name: d:\delphi5\drbob\twmlprod
Search path: $(DELPHI)\Lib;$(DELPHI)\Bin;$(DELPHI)\Imports;$(

➤ *Figure 2: New TWMLDataSeTTableProducer Component Dialog.*

This results in two overridden methods (TableHeader and Content) in the TWMLDataSetTableProducer component, and one new function (WMLTable). A final new function called EnCode is needed because WML is more sensitive to the use of special characters such as quotes, apostrophes, ampersands, less than characters, greater than characters and soft hyphens. We need to replace these with the WML codes, or character entities (respectively with &quot; &apos; &amp; &lt; &gt; &shy; and &nbsp;). This ensures that the generated field values are still compatible with WML and will not produce error messages (a lonely & will surely do otherwise).

After installation, you'll find the new TWMLDataSetTableProducer on your component palette, ready to be used in WebBroker WAP applications (note that the output is now of less use in 'traditional' HTML web applications, although web browsers tend to be more forgiving than WAP phones).

A nice side effect of the Property Editor for the Columns property of the TWMLDataSetTableProducer is that it has a preview page that shows how the output will look. At that point, the Object Inspector shows a number of properties that are useful when generating HTML output (like bgcolor, border, etc) which are not used by TWMLDataSetTableProducer. And since the Property Editor still uses a real call to the underlying TWMLDataSetTableProducer.Content method to preview the output, you will immediately see that setting these properties at design-time has no effect on the output. So even if you see the output in an HTML browser (preview) and not in a WAP phone, it still shows only the effects for the properties that are indeed supported by WML.

Of course, in time one could even enhance the Property Editor for the TWMLDataSetTableProducer to use a WAP phone simulation as the preview window, but that's something for another day.

Final Demonstration

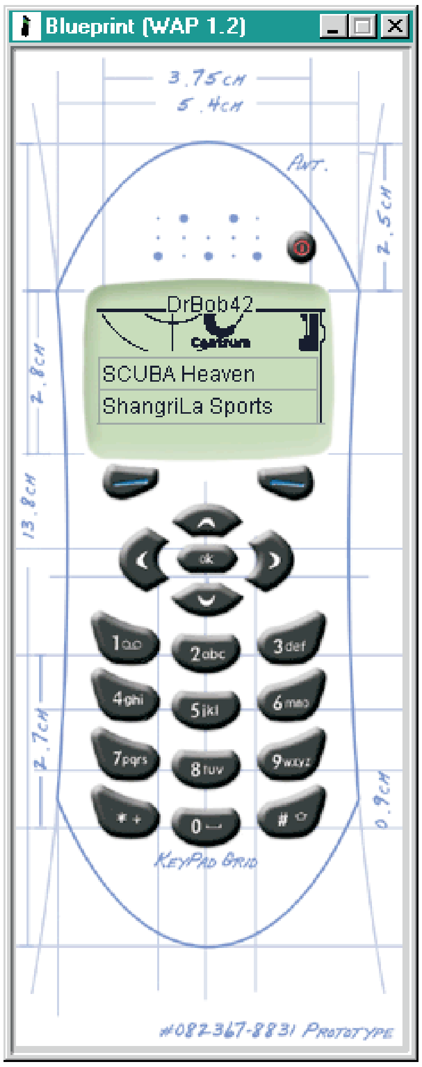
As a final demonstration, let's write a WebBroker application that uses a TWMLDataSetTableProducer to produce a WML table as well as a dynamic WBMP image. We need to WebItemActions; one default action to produce the initial WML and to include

➤ *Listing 2: TWMLDataSetTableProducer component.*

```
unit twmlprod;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, HTTPApp, DB, DBWeb;
type
  TWMLDataSetTableProducer = class(TDataSetTableProducer)
  public
    function TableHeader: string; override;
    function Content: string; override;
  end;
procedure Register;
implementation
const
  StartRow = '<tr>';
  EndRow = '</tr>';
function EnCode(Str: String): String;
{ Convert memo contents to single line XML }
var
  i: Integer;
begin
  for i:=Length(Str) downto 1 do begin
    case Str[i] of
    '"': begin
           Insert('&quot;',Str,i+1);
           Delete(Str,i,1)
         end;
   '''': begin
           Insert('&apos;',Str,i+1);
           Delete(Str,i,1)
         end;
    '&': begin
           Insert('&amp;',Str,i+1);
           Delete(Str,i,1)
         end;
    '<': begin
           Insert('&lt;',Str,i+1);
           Delete(Str,i,1)
         end;
    '>': begin
           Insert('&gt;',Str,i+1);
           Delete(Str,i,1)
         end;
    '-': begin
           Insert('&shy;',Str,i+1);
           Delete(Str,i,1)
         end;
    else
      if (Ord(Str[i]) in [1..31]) then begin
        Insert('&#'+IntToStr(Ord(Str[i]))+';',Str,i+1);
        Delete(Str,i,1)
      end else
        if Str[i] = #0 then Delete(Str,i,1)
    end
  end;
  Result := Str
end {EnCode};
function WMLTable(DataSet: TDataSet; DataSetHandler:
  TWMLDataSetTableProducer; MaxRows: Integer): string;
var
  I, J: Integer;
  DisplayText: string;
  Field: TField;
  Column: THTMLTableColumn;
begin
  Result := DataSetHandler.TableHeader + #13#10;
  if DataSet.State = dsBrowse then begin
    J := 1;
    while (MaxRows <> 0) and not DataSet.EOF do begin
      Result := Result + StartRow;
      for I := 0 to DataSetHandler.Columns.Count-1 do begin
        Column := DataSetHandler.Columns[I];
        Field := Column.Field;
        if Field <> nil then
          DisplayText := EnCode(Field.DisplayText)
        else
          DisplayText := '';
        with Column do
          Result := Result + DataSetHandler.FormatCell(J, I,
            DisplayText, ‘td’, '', Align, VAlign, '');
      end;
      Result := Result + EndRow + #13#10;
      DataSet.Next;
      Dec(MaxRows);
      Inc(J);
    end;
  end;
  Result := Result + '</table>';
end;
function TWMLDataSetTableProducer.Content: string;
begin
  Result := '';
  if DataSet <> nil then begin
    if DataSet.Active and (Columns.Count = 0) then
      LayoutChanged;
    if DoCreateContent then
      Result := Header.Text + WMLTable(DataSet, Self,
        MaxRows) + Footer.Text;
  end;
end;
function TWMLDataSetTableProducer.TableHeader: string;
begin
  Result := '<table';
  with TableAttributes do begin
    if Width > 0 then
      Result := Format('%s columns="%d"',
        [Result, Columns.Count]);
    if Custom <> '' then
      Result := Format('%s %s', [Result, Custom]);
  end;
  Result := Result + '>';
end;
procedure Register;
begin
  RegisterComponents('DrBob42', [TWMLDataSetTableProducer]);
end;
end.
```

```
procedure TWebModule1.WebModule1WebActionItem2Action(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="DrBob42" title="DrBob42">'#13#10 +
    '<p>'#13#10 +
    '<img src="http://192.168.92.201/cgi-bin/tdm65.exe/image" alt="logo"/>' +
    '</p>'#13#10'<p>' +
     WMLDataSetTableProducer1.Content + #13#10 +
    '</p>'#13#10 +
    '</card>'#13#10 +
    '</wml>';
end;
```

➤ *Listing 3: Final WML/WBMP Example.*



➤ *Figure 3*

the TWMLDataSetTableProducer's Content output, and a /image action item to produce the dynamic image (as we saw in Listing 1).

First, drop a TTable component on your web module, and make it point to something (like the Customer.db table from DBDEMOS). Now, before you continue, it's important to remember that a WAP phone has only a small display, so you don't want to show more than a few fields. In my example, I even limit myself to one field (the company name), and I urge you to experiment with WML tables before using them in a real-world WAP application. Anyway, once your table points to a dataset, you can drop on a TWMLDataSetTableProducer component, point its DataSet property to the table and click on the ellipsis (...) next to the Columns property to visually 'design' your output.

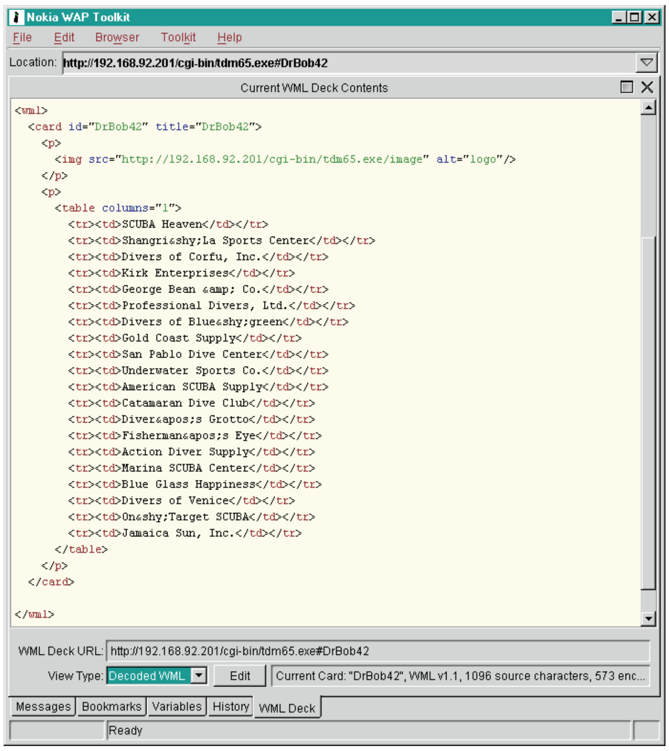
Finally, since we've already seen the code for the /image OnAction item, we only need to focus on the initial OnAction event handler, which can be coded as shown in Listing 3.

As you can see, the call to tdm65.exe/image is used to dynamically produce a WBMP image, which is followed by the WML table as produced by WMLDataSetTable-Producer1.Content. The output can be seen in Figures 3 and 4.

Next Time

It's been a while since I've covered CORBA or the VisiBroker for Delphi add-in tool in these pages. And by the time you read this column, a new version of VisiBroker 3.3 for Delphi 5 should be available from Borland (most probably as a *paid-for* add-in tool for Delphi 5 Enterprise). Next time in *Under Construction* we will see how this new VisiBroker for Delphi has matured, and what the new IDL-2-PAS can produce for us. Support for CORBA exceptions may no longer be limited to client-side only, and we'll be working with true server skeletons and client stubs. Finally, CORBA the way it was meant to be with Delphi! All this and more next month, *so stay tuned...*

➤ *Figure 4*


Nokia WAP Toolkit
File Edit Browser Toolkit Help
Current WML Deck Contents
Messages Bookmarks Variables History WML Deck
Ready

Location: http://192.168.92.201/cgi-bin/tdm65.exe#DrBob42

```
<wml>
  <card id="DrBob42" title="DrBob42">
    <p>
      <img src="http://192.168.92.201/cgi-bin/tdm65.exe/image" alt="logo"/>
    </p>
    <p>
      <table columns="1">
        <tr><td>SCUBA Heaven</td></tr>
        <tr><td>Shangri&shy;La Sports Center</td></tr>
        <tr><td>Divers of Corfu, Inc.</td></tr>
        <tr><td>Kirk Enterprises</td></tr>
        <tr><td>George Bean &amp; Co.</td></tr>
        <tr><td>Professional Divers, Ltd.</td></tr>
        <tr><td>Divers of Blue&shy;green</td></tr>
        <tr><td>Gold Coast Supply</td></tr>
        <tr><td>San Pablo Dive Center</td></tr>
        <tr><td>Underwater Sports Co.</td></tr>
        <tr><td>American SCUBA Supply</td></tr>
        <tr><td>Catamaran Dive Club</td></tr>
        <tr><td>Diver&apos;s Grotto</td></tr>
        <tr><td>Fisherman&apos;s Eye</td></tr>
        <tr><td>Action Diver Supply</td></tr>
        <tr><td>Marina SCUBA Center</td></tr>
        <tr><td>Blue Glass Happiness</td></tr>
        <tr><td>Divers of Venice</td></tr>
        <tr><td>On&shy;Target SCUBA</td></tr>
        <tr><td>Jamaica Sun, Inc.</td></tr>
      </table>
    </p>
  </card>

</wml>
```

WML Deck URL: http://192.168.92.201/cgi-bin/tdm65.exe#DrBob42
View Type: Decoded WML   Edit   Current Card: "DrBob42", WML v1.1, 1096 source characters, 573 enc...

Acknowledgements

I want to thank my Everest DOC colleague Arnim Mulder for his welcome assistance and suggestions while playing with WAP, WML and WBMPs.

Bob Swart (aka Dr.Bob, visit www.drbob42.com) is an @-consultant for Everest and also co-founder of the Delphi OplossingsCentrum. Bob is a freelance technical author and a frequent speaker at Delphi-related events all over the world.