Techniques for an Enhanced WAP Graphical User Interface

Alaistair Deacon, Real Time Engineering Ltd

In this session, we will take a look at some techniques for delivering a graphical user interface (GUI) within the limited common facilities provided by the current range of WAP 1.1 compliant browsers and handsets.

We will examine the current crop of browsers, and gain an understanding of how they render WML, especially images. From this understanding of their capabilities, we can create a framework for delivering a WML deck — built from a number of separate images — that will simulate a GUI when rendered by the browser. Finally, we'll turn this static GUI into a dynamic one through the server-side creation of graphics at runtime.

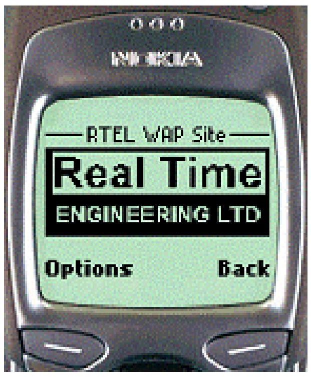
This session will explore the techniques required to create this dynamic environment, and presents a WBMP COM object for manipulating and creating dynamic WBMPs on the server.

Introduction

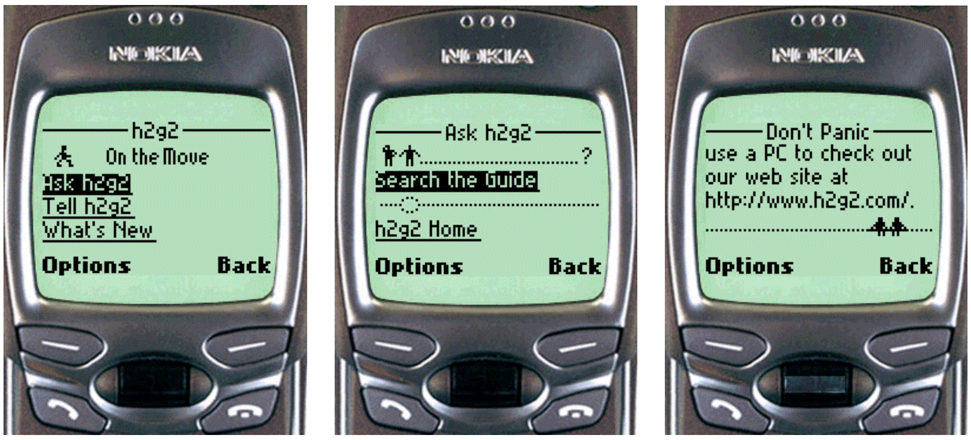
Most of us (and our clients) are used to the rich graphical environments that are normally provided by the Web, using all kinds of graphics, frames, and tools like Flash. However, while WAP does provide support for displaying graphics within a WML deck, this is currently limited to 'Type 0' WBMPs: 1-bit, monochrome bitmaps.

When first exposed to the realities of WAP, the graphical limitations have come as a bit of a shock to many of our clients, especially when they've heard all the advertisers' promises of WAP being the "Internet in your hand". But then, what have developers *done* with the graphical capabilities of WAP so far? The almost universal answer to this is... not much! Almost all current WAP sites are heavily text-based, and few have slick and consistent navigation.

We also get caught up with our clients' marketing departments, who want to ensure that the corporate image and style they have spent so much time developing in their web site is also conveyed within the WAP equivalent. To date, our solution has been to create splash screens on entry to major sub-sections of the WAP sites we've developed. My own company's site is no exception:



A few sites (such as the Hitchhiker's Guide to the Galaxy at http://wap.h2g2.com) have used some more innovative graphics. They use inline graphics that are approximately the same height as a line of text to create interesting page headers and breaks:



use a PC to check out our web site at http://www.h2g2.com/.

In addition, some sites have included basic graphs for things like share prices, etc., but these are normally created at fixed intervals and simply dumped onto the WAP server.

What's *really* required is a GUI that delivers dynamic information within the WAP environment. Furthermore, it must render this information on anything from the most basic WAP browser (such as the one on the Nokia 7110) to the more advanced browsers available on devices such as the Ericsson R380 and the Palm PDA.

The example described in this paper is an airline check-in application. If you're a frequent flyer, you'll be used to electronic airline tickets and self-service check-in kiosks at airport terminals — the ones that allow you to insert your airline loyalty card and select your seat preference on screen. The less organized among us often resort to calling the airline's call center from our mobile phone while rushing to the airport at the last minute. The aim of any WAP check-in application should be to provide the same rich graphical environment as the self-service kiosk.

WAP Browser Graphical Rendering

In order to attack the problem, we need to gain a deeper understanding of the facilities defined in WAP 1.1 for rendering images, and the extent to which these facilities are implemented by the current generation of WAP browsers. Let's start with the first task.

In WML, we can include an image within a card by using the <img> tag:

```
<img src="rtel.wbmp" alt="Real Time Engineering Limited"/>
```

Version 1.1 of the WML Specification includes a number of formatting options that can be used with the <img> tag. First, any image should follow the alignment rules of the paragraph containing it:

```
<p align="right">
   <img src="partcldy.wbmp" alt="partcldy.wbmp"/>
</p>
```

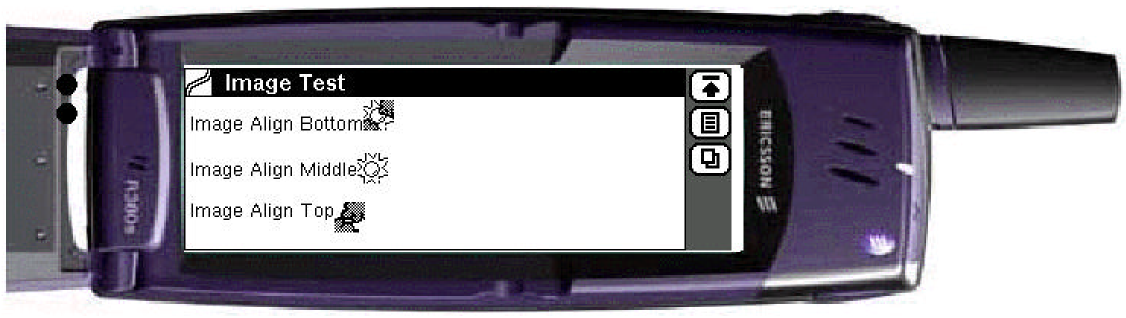
Additionally, the <img> tag has formatting attributes of its own, and these include:

```
align={bottom|middle|top}
```

This attribute specifies image alignment within the text flow, with respect to the current insertion point. `align` has three possible values:

- `bottom` — The bottom of the image is vertically aligned with the current baseline. This is the default value.
- `middle` — The center of the image is vertically aligned with the center of the current text line.
- `top` — The top of the image is vertically aligned with the top of the current text line.

These alignment attributes are shown on the Ericsson R380 below.



The following attributes specify the amount of whitespace to be inserted to the left and right (hspace) and above and below (vspace) an image or object.

```
vspace=length
hspace=length
```

However, anyone who starts to experiment with using any of the above attributes in different WAP browsers will soon be disappointed, because support for them is not mandatory and they are commonly ignored. Sadly, we're in a situation similar to the days of the Internet browser wars. The different browsers render markup and graphics differently, and for developers this means testing our applications on all of them and programming to the lowest common denominator; or attempting to detect the type of WAP browser being used, and tailoring the content appropriately.

The different WAP browsers provide different levels of support for rendering a WML deck, and these variations further extend to how the browser controls the scrolling of graphics. When developing, then, the questions you should ask yourself include:

- Does the browser support alignment rules?
- Does the browser support inline text and graphics?
- Does the browser support multiple graphics in a single line?
- Does the browser support scrolling of graphics within the window?
- How does the browser render a <br/> tag between graphics?
- How does the browser render <p> and </p> tags around a graphic?

Once we're able to define these capabilities within the range of browsers we are targeting, we can gain an understanding of what's required to build our GUI.

A set of WML cards that can be used as a basis to test the rendering capabilities of the target browser is provided in Example 1 on the CD accompanying the conference. A summary of my findings is given in the table below:

	Nokia 7110	**Ericsson R320**	**Ericsson R380 Simulator**	**Phone.com UP.Browser**	**Mitsubishi TRIUM**
*Screen resolution available for browser	96x44	101x52	310x100	Phone dependent	Estimated at 96x40
Invert bitmaps	No	No	No	Yes	No
Placeholder while downloading	No	Yes	Yes	No	No
Smooth scrolling through graphics larger than the display area	No	Yes	Yes	No	Yes
Inline text and bitmaps	No	No	Yes	Yes	Yes
Multiple bitmaps in the same line	No	No	Yes	Yes	Yes
<br/> tag between graphics	New Line (large gap)	New Line (with gap)	New Line (no gap)	New Line (no gap)	New Line (no gap)
<p></p> around a graphic	New Line (slight gap)	New Line (with gap)	New Line (with gap)	New Line (no gap)	New Line (no gap)
Text right align	No	Yes	Yes	Yes	Yes
Text left align	No	Yes	Yes	Yes	Yes
Text center align	No	Yes	Yes	Yes	Yes
Text big style	No	Appears bold	Yes	No	Yes
Text emphasis style	No	Appears bold	Appears bold	Appears bold	Appears big
Text italics style	No	No	Appears bold	No	Appears bold
Text small style	No	Yes	Yes	No	Yes
Text strong style	No	Appears bold	Appears bold	Appears bold	Appears big
Text underline style	No	Yes	Appears bold	Appears bold	No
Image left align	No	No	Yes	No	No
Image right align	No	No	Yes	No	No
Image center align	Yes	Yes	Yes	No	Yes

**The actual screen resolution may be larger, with an area reserved for the softkey labeling.*

The Service Developer's Guide for the Nokia 7110 — Nokia's own documentation for the capabilities of the browser on its 7110 mobile phone — makes interesting reading. (This information is available from Nokia, and comes with the Nokia WAP Development Kit 1.3 Beta.) Of particular interest to this article is the description of the graphics rendering capabilities. A few important points to highlight are:

The Nokia 7110 supports Wireless Bitmap graphics. The maximum size of the graphics area is 96x44 pixels.

Images fitting in the graphics area are centered. Images that are larger than the display are left-aligned and truncated from the right-hand part. Images which are taller than the display are top-aligned, and the bottom part of the display is truncated.

There cannot be text next to an image; the image will always start its own line.

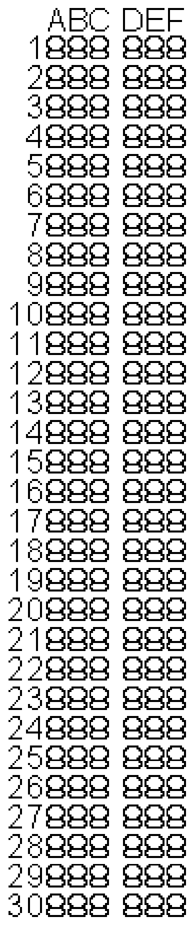
The user can scroll an image line-by-line, but when the image is scrolled to the top of the display, the entire image will disappear with the next scroll.

In the example application that follows, I will demonstrate that even with such a limited graphical capability, we can combine multiple graphics within a single WML card to create a scrollable graphical environment.

Delivery of the WAP Graphical Environment to the Browser

Returning to the goal of this article, the aim is to create a dynamic, graphical representation of an airline self-service check-in kiosk, complete with a seating plan.

Given an unlimited graphical environment, we would typically want to deliver to the browser an image like the one below, allowing the user to scroll through the aircraft cabin:

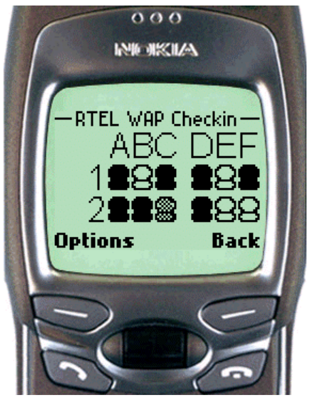


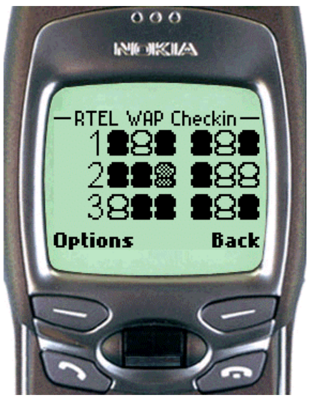
Despite all the limitations of microbrowsers, we can deliver this environment by understanding the capabilities of our target browser. Looking at the table of the browser capabilities, it can be seen that if we can deliver this environment to the Nokia 7110, all the other browsers will be able to support it.

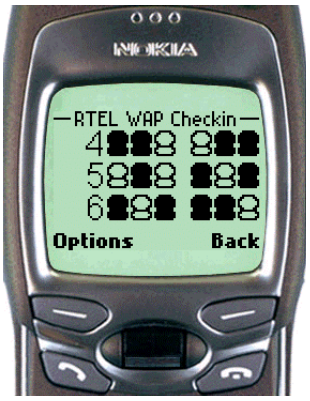
By breaking up our cabin layout graphic into manageable chunks and including them in a single WML card, we can deliver this environment to the browser. Example 2 on the CD gives an example of this: we have broken up the cabin layout into chunks of 96x44 pixels, and stacked them vertically in the WML card:

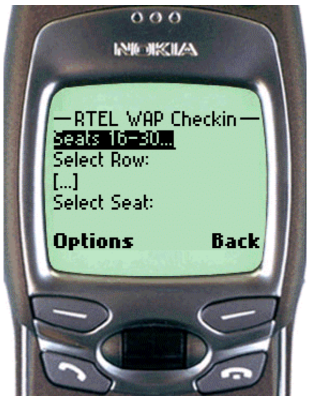
```
<card id="card1" title="RTEL WAP Checkin">

   <p align="left">
      <img src="header.wbmp" alt=""/>
   </p>
   <p align="left">
      <img src="seats0.wbmp" alt="seat graphic"/>
   </p>
   <p align="left">
      <img src="seats1.wbmp" alt="seat graphic"/>
   </p>
   <p align="left">
      <img src="seats2.wbmp" alt="seat graphic"/>
   </p>
   <p align="left">
      <img src="seats3.wbmp" alt="seat graphic"/>
   </p>
   <p align="left">
      <img src="seats4.wbmp" alt="seat graphic"/>
   </p>
   ...
   <p align="left">
      <a href="checkin2.wml">Seats 16-30...</a>
   </p>
   <p align="left">
      Select Row: <input format="*N" name="row" title="Row"/>
   </p>
   <p align="left">
      Select Seat:
      <select name="seat" multiple="false" ivalue="1">
         <option value="A">A</option>
         <option value="B">B</option>
         <option value="C">C</option>
         <option value="D">D</option>
         <option value="E">E</option>
         <option value="F">F</option>
      </select>
   </p>
   <p align="left">
      <a href="#confirm">Check In...</a><br/>
      <do type="accept" label="Check In...">
         <go href="#confirm"/>
      </do>
   </p>

</card>
```

(Note that the file `checkin2.wml` contains a similar card for seats in rows 16 to 30.) When this card is rendered by the 7110, it displays the graphics chunks, as shown below. As the user scrolls through the deck, each graphic chunk is displayed in turn, giving the impression that they're scrolling through a much larger graphic.



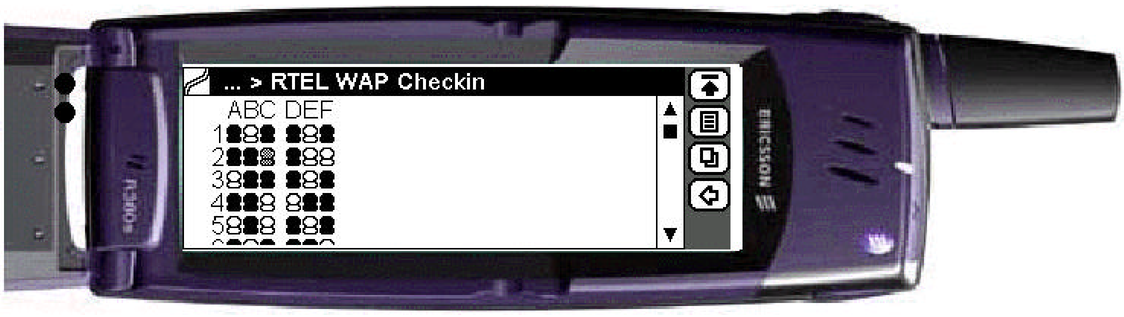






Obviously, this technique doesn't give us an *unlimited* scrolling size, as there are limitations on the device's memory capacity. In experiments, however, I've found that it's possible to display seating arrangements of over twenty rows of six seats.

As the application only needs limited browser support, it should operate on most browsers, as shown below on the Ericsson R380 and the Phone.com UP.Simulator 4.0.



Now that we have a method for displaying what appears to be a large scrolling graphic to all browsers, we need to look at how to create this page dynamically.

The WBMP File Structure

The WBMP format is a simple binary file format used to store bitmaps. Details of this file type are available in the Wireless Application Environment (WAE) Specification, which is available for download from the WAP Forum (http://www.wapforum.org).

The WBMP format is extensible, and the type of a particular bitmap is defined in the file header. (This will allow for future expansion to include full color, animations, etc.) At present, however, only 'Type 0' is defined, and it is unlikely that this will change in the near future (unless devices such as the color Palm make an early move to provide colorful WAP browsers).

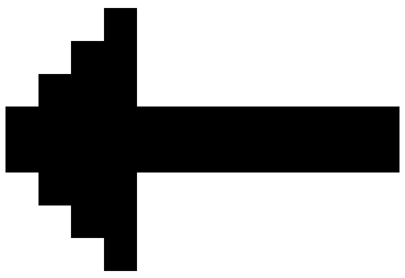
WBMP is a simple format that defines the size of the bitmap in terms of width and height, followed by a number of bytes containing the image data.

The width and height of the bitmap are given as multi-byte integers. These integers consist of a number of octets, the most significant bit of each being a continuation flag (that is, a value 1 means there are more bytes to follow, and a value 0 means that it is the last byte). The remaining seven bits give the scalar value.

The WBMP image data is organized in pixel rows, which are represented by a sequence of octets. One bit represents the pixel intensity, with white being 1 and black being 0. In a situation where the row length is not divisible by 8, the encoding of the next row must start at the beginning of the next octet, and all unused bits must be set to zero. This is shown in the figure below — a bitmap of 12 pixels wide by 8 pixels high is encoded as 16 bytes like this (where the 'used' bits are shaded):

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 3...															
Byte 5...															
Byte 7...															
Byte 9...															
Byte 11...															
Byte 13...															
Byte 15...															

Encoding the following 12x8 bitmap would give the following image data:



1	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0
1	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0

It's a relatively easy task to write a server-side application that can generate the required graphics in WBMP format dynamically at runtime. As you'll see later, I've provided a COM object that provides a simple interface for manipulating these WBMPs.

Creation of WBMP Templates

If we know the desired format of our final graphical layout, what's the best way to go about creating it dynamically? The technique I've developed is to create 'WBMP templates'.

We start with a template of the desired final output, including all text and logos. This can be created easily in Paint Shop Pro, using the WBMP plug-in available from http://www.rcp.co.uk/distributed/Downloads, or by using any similar package. The template of the aircraft cabin, already shown, includes all the row numbering and nothing but 'empty' (white) seats.

Next, we prepare any additional graphics that are required. In this example, these are a booked seat (black) and the passenger's assigned seat (gray, or rather a mixture of black and white pixels), as shown below:

Once these templates have been created, we use a server-side application to assemble them in a way that represents the dynamic seating plan.

Assembly of WBMP Templates

The assembly of the WBMP templates could be performed by any of the main server-side application languages. Being a Microsoft Solution Developer, however, and since our database was running on SQL Server, we decided to create a COM object to manipulate the WBMPs in Visual C++, and to use Visual Basic and VBScript within ASP to create our dynamic solution.

A copy of the WBMP COM object has been included in the supporting material.

The methods provided by the WBMP COM object are given in the following table:

Method	**Description**
`WBMP.CreateWBMP(NewHeight, NewWidth)`	Creates a blank WBMP of the desired size
`WBMP.LoadWBMP(Filename)`	Loads a WBMP from file
`WBMP.SaveWBMP(Filename)`	Saves a WBMP to file
`WBMP.SaveWBMPSection(X1, Y1, X2, Y2, Filename)`	Clips and saves a WBMP subsection
`WBMP.PlaceWBMP(WBMP, X1, Y1, [X2], [Y2], [Width], [Height])`	Places one WBMP on top of another WBMP at position X1, Y1 (optionally clips from X2, Y2, Width, and Height on source WBMP)
`WBMP.Height()`	Returns the height of the WBMP
`WBMP.Width()`	Returns the width of the WBMP
`WBMP.Bitmap(X, Y)`	Sets or returns the value of an individual pixel

WBMP.PlaceWBMP() merits further explanation. If I simply need to place a small source WBMP (say a single seat) on top of a larger template WBMP at location x1, y1, I would use the function call:

```
      wbmpTemplate.PlaceWBMP(wbmpSource, x1, y1)
```

However, if I wanted to place a *section* of a large source WBMP on top of the template WBMP, the parameters x2, y2, width, height would be used to specify the section of the source WBMP to be copied.

```
      wbmpTemplate.PlaceWBMP(wbmpSource, x1, y1, x2, y2, width, height)
```

With this suite of methods, we can construct the dynamic WBMP. This is achieved in four steps. First, load the basic WBMP template for the seating plan, and the additional WBMP graphics to be placed on top:

```
Dim wbmpTemplate as New WBMP
Dim wbmpSeatOccupied as New WBMP
Dim wbmpSeatReserved as New WBMP

wbmpTemplate.LoadWBMP("template.wbmp")
wbmpSeatOccupied.LoadWBMP("occupied.wbmp")
wbmpSeatReserved.LoadWBMP("reserved.wbmp")
```

Next, we need to open the database to construct the current seating listing:

```
' Open database
' for each occupied seat
'    Get the location of the occupied seat in x and y pixels
'    Place the occupied seats on the template
     wbmpTemplate.PlaceWBMP(wbmpSeatOccupied, x, y)
' end for

' Get reserved seat from the database
' Get the location of the reserved seat in x and y pixels
' Place the reserved seat on the template
  wbmpTemplate.PlaceWBMP(wbmpSeatReserved, x, y)
```

Now that we have created our completed seating plan, we need to save it in individual chunks to be delivered to the phone:

```
' for each chunk of the template
     wbmpTemplate.SaveWBMPSection(x1, y1, x2, y2, "seats" & index & ".wml")
' end for
```

The last stage is to tidy up our created objects:

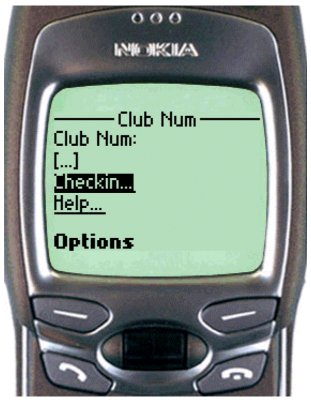
```
Set wbmpTemplate = Nothing
Set wbmpSeatOccupied = Nothing
Set wbmpSeatReserved = Nothing
```

A sample Visual Basic application is provided in the supporting material to demonstrate this code in action.

Example Seat Booking Application

To complete the seat booking application, all that is required is to integrate these techniques into a complete set of WML cards. The first card should request the user's frequent flyer card number, as shown in the figure that follows this listing:

```
<card id="card1" ontimer="#club" title="RTEL WAP Checkin">
   <timer value="25"/>

   <p align="center">
      <img src="rtel.wbmp" alt="Real Time Engineering"/><br/>
      <big><b>WAP Checkin</b></big><br/>
   </p>
</card>

<card id="club" title="Club Num" newcontext="true">
   <p>
      WAP Checkin Demo.<br/>
      (c)Real Time Engineering.<br/>
      Club Num: <input format="*N" name="exec" title="Exec Num"/>
   </p>
   <p>
      <a href="#handbaggage">Checkin...</a>
   </p>
   <p>
      <a href="#card1_help">Help...</a><br/>
         <do type="accept" label="Checkin">
            <go href="#handbaggage"/>
         </do>
         <do type="help" label="Help">
            <go href="#card1_help"/>
         </do>
   </p>
</card>

<card id="handbaggage" ontimer="checkin1.wml"
      title="RTEL WAP Checkin" newcontext="false">
   <timer value="25"/>
   <p align="center">
      Handbaggage Passengers Only!<br/>
   </p>
</card>
```



The frequent flyer card number would be passed to an ASP script that checks the booking details and then creates the WBMP seating plan, as described above. The same script would then generate the WML card.

The customer can then browse through the available seats, and select either the assigned seat or an alternative. This selection is then submitted to another ASP script that will confirm the check-in booking.

> *Details of the BackOffice integration of ASP are not covered in this session. Check out resources such as http://www.asptoday.com and other sessions at this conference for descriptions of WAP development using ASP, as well as* Professional WAP *from Wrox.*

Further Work

There are a couple of opportunities for the further development of this application. First, there is an issue of concurrency. In the WML card given in Example 2, the individual WBMPs are named `seats1.wbmp`, `seats2.wbmp`, etc. In a live application, we would need to generate a unique set of WBMPs for each user checking in. Therefore, the script generating the dynamic WBMPs should use an index counter or a GUID as a root filename for the dynamically created WBMPs. The script should then create a deck similar to the one in Example 2, except that the references to the images should point to the uniquely created WBMPs for that session. A background application could be created to delete the used WBMPs after a suitable interval.

An alternative solution to this problem would be to create ASP scripts that return the raw WBMP data directly. This can be achieved by creating a return type of

```
Response.ContentType = "image/vnd.wap.wbmp"
```

in the ASP header, and streaming out the raw WBMP data directly from the ASP script. However this is outside the scope of this presentation.

Finally, you may want to tailor your ASP output by checking the type of WAP browser being used by the agent. Check out Wei Meng Lee's talk on XML and XSLT, and *Professional WAP*, for tips on how to do this.