Creating a Dynamic WAP Application using ASP: The Mobile Personal Assistant

Wei Meng Lee, Ngee Ann Polytechnic, Singapore

Microsoft Active Server Pages (ASP) is a viable technology for developing dynamic web content. Its popularity can be seen from the many web sites sporting documents with the .asp extension. Because ASP is a server-side technology, it is well suited to creating dynamic WAP applications, especially in the area of database access.

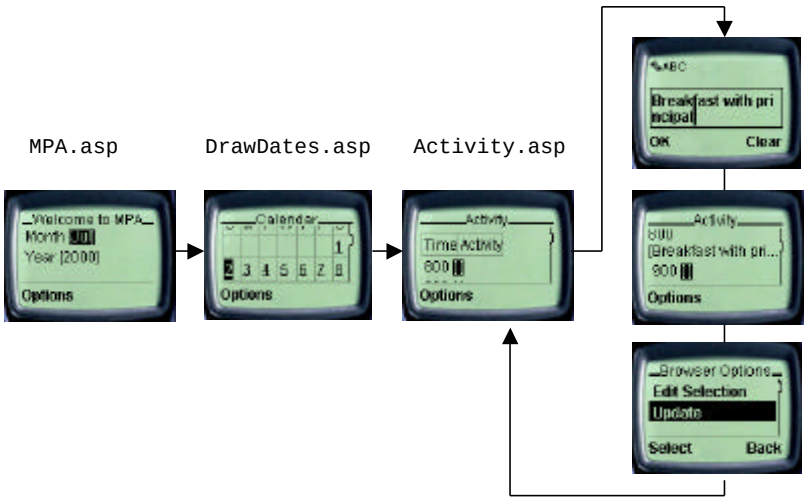
In the first part of this paper, I will look at how we can use ASP to create a dynamic WAP application: the Mobile Personal Assistant (MPA). Using the MPA, users are able to view a calendar on their WAP-enabled phones, and select a date and time at which they can schedule activities.

In the second part of the paper, I will highlight various techniques for creating dynamic WAP applications. Issues like caching, detecting WAP devices, and common errors associated with ASP will be covered.

The MPA Application

As I mentioned in the introduction, the sample application that we will be building in this session is a Mobile Personal Assistant (MPA). The aim of this is to allow users to have a mobile time scheduler that enables them to check on their plans wherever they are.

Application Overview



As can be seen from the above diagram, there are three decks involved in this application. They are:

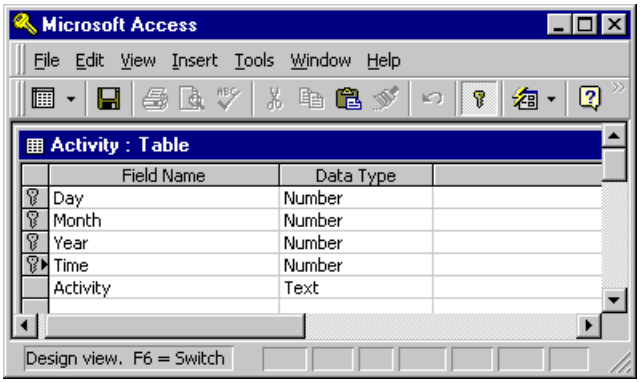
- MPA.asp
- DrawDates.asp
- Activity.asp

I will explain what these decks are for when we use them in the following sections.

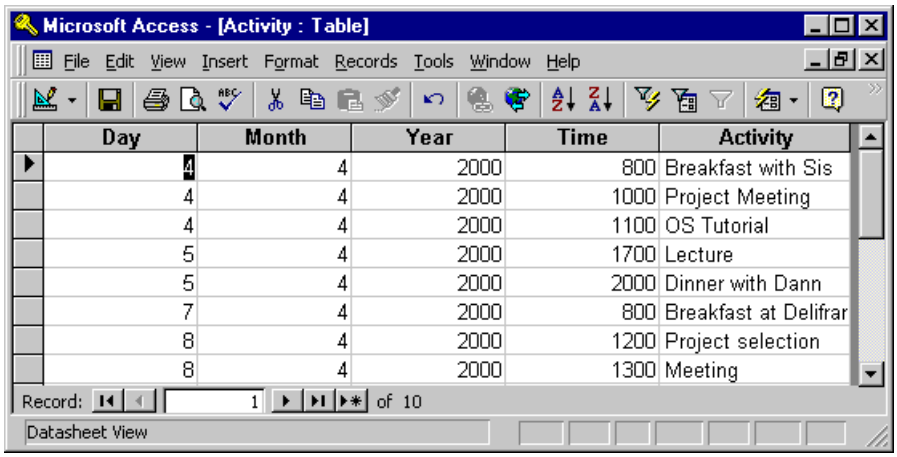
In this paper, I have assumed that the reader is familiar with ASP. I will not attempt to explain the workings of ASP in detail. Rather, I will explain the concepts that are important in getting the sample application in this paper to work.

> *For a good tutorial on ASP, you could refer to either Beginning Active Server Pages 3.0 or Beginning ASP Databases, both published by Wrox Press.*

In implementing this application, the technologies I'll use are ASP and ActiveX Data Objects (ADO). Let's first take a look at the database:

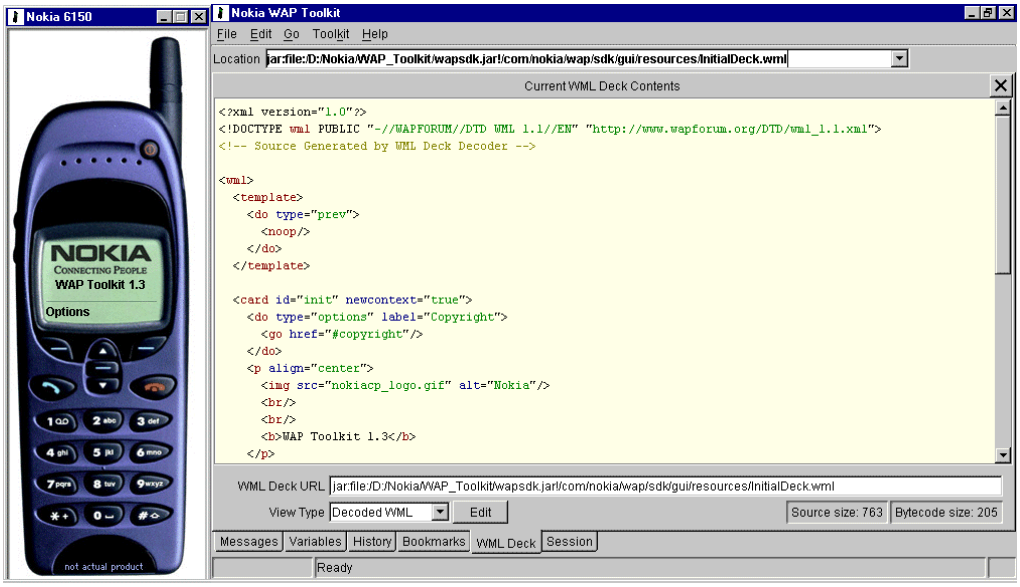


This Microsoft Access database (`PIM.mdb`) contains a single table for storing the activities of the user. A typical schedule looks like this:



WAP Browser

To develop the application, I used the Nokia WAP Toolkit 1.3. The Nokia WAP Toolkit can be downloaded from http://www.nokia.com/wap at no charge.



Creating a Calendar

To enable the user to locate a day and month in their schedule easily, I need to create a calendar on their screen. This is achieved by the code in `MPA.asp`, which you'll find in the accompanying material, and looks like this:

Let's now dissect the code to see how this interface is created. First, there's the standard header for WML content:

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                     "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
```

Next comes the card that enables the user to select the month and year, as follows:

```
   <card id="card1" title="Welcome to MPA">
      <p>
         Month <select name="Month">
            <option value="1">Jan</option>
            <option value="2">Feb</option>
            <option value="3">Mar</option>
            <option value="4">Apr</option>
            <option value="5">May</option>
            <option value="6">Jun</option>
            <option value="7">Jul</option>
            <option value="8">Aug</option>
            <option value="9">Sep</option>
            <option value="10">Oct</option>
            <option value="11">Nov</option>
            <option value="12">Dec</option>
         </select>
```

```
         Year <select name="Year">
            <option value="2000">2000</option>
            <option value="2001">2001</option>
            <option value="2002">2002</option>
            <option value="2003">2003</option>
         </select>
```

Once the month and the year have been selected, they are sent to the `DrawDates.asp` document using the `GET` method. A `<postfield>` element is used to accomplish this:

```
         <do type="accept" label="View Calendar">
            <go href="DrawDates.asp#card2" method="get">
               <postfield name="Month" value="$Month" />
               <postfield name="Year" value="$Year" />
            </go>
         </do>
      </p>
   </card>
</wml>
```

The `href` attribute instructs the WAP device to load up `card2` in `drawdates.asp`.

The DrawDates.asp Document

I need to draw the calendar on the user's screen, so that they can select the day in which to enter their schedule. The `generateDate()` function is used to generate a table containing all the days in a particular month and year:

```
   <%
   Function generateDate(mm, yy)
      '-------------------------------------------------------
      '--- Determine the 1st of the month starts on which day
      dateStr = mm & "/" & "7" & "/" & yy
      startday = Weekday(dateStr)

      '--- The calendar will start from Sunday(1), Monday(2)...
      outStr = "<table columns='7'>"
      outStr = outStr & _
               "<tr><td>S</td><td>M</td><td>T</td><td>W</td>" & _
               "<td>T</td><td>F</td><td>S</td></tr>"
      outStr = outStr & "<tr>"

      '--- Writes the filler
      Counter = 1
      While Counter < startday
         Counter = Counter + 1
         outStr = outStr & "<td></td>"
      Wend

      For i = 1 To 31
         dateStr = mm & "/" & i & "/" & yy
         If isDate(dateStr) Then
            outStr = outStr & "<td>"
            outStr = outStr & "<a href='Activity.asp?dd=" & _
                     i & "&amp;mm=" & mm & "&amp;yy=" & _
                     yy & "'>" & i & "</a>"
            outStr = outStr & "</td>"
            If Counter Mod 7 = 0 Then
               outStr = outStr & "</tr><tr>"
            End If
         Else
            Exit For
         End If
         Counter = Counter + 1
      Next

      outStr = outStr & "</tr></table>"
      generateDate = outStr
   End Function
```

```
'------------------------------------------------------------
'--- Draws the calendar
Dim mm
Dim yy
mm = Request.QueryString("Month")
yy = Request.QueryString("Year")
%>

<card id="card2" title="Calendar">
   <p>
      <%
      If mm="" Then
         mm = month(date)
         yy = year(date)
      End If

      Response.Write "Date :" & mm & "/" & yy
      Response.Write generateDate(mm,yy)
      %>
      <do type="accept" label="Reselect Date">
         <go href="MPA.asp#card1" method="get" />
      </do>
   </p>
</card>
```

Note that I'm creating links for each and every day, all of which point to the document `Activity.asp`. I also pass the values of the day, month, and year.

Dealing with Ampersands

In the conventional HTML anchor tag, if I want to pass multiple (name, value) pairs to a document, I use something like this:

```
Activity.asp?dd=1&mm=2000
```

In WML, however, the & character has a special meaning, and we can't use it to separate pairs. That's why we find ourselves writing code like this, which uses `&` to represent the & character:

```
outStr = outStr & "<a href='Activity.asp?dd=" & i & _
         "&amp;mm=" & mm & "&amp;yy=" & yy & "'>" & i & "</a>"
```

The Activity.asp Document

Once the user has selected a day, they can click on the Follow Link option to retrieve the activity page:

Without further ado, let's take a look at the code that generates this card:

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<% Response.Expires=0 %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                     "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
   <%
   Dim dd
   Dim mm
   Dim yy
   Dim conn, rs
   dd = Request.QueryString("dd")
   mm = Request.QueryString("mm")
   yy = Request.QueryString("yy")

   Set conn = Server.CreateObject("ADODB.Connection")
   Set rs   = Server.CreateObject("ADODB.Recordset")

   '===Connect to the database===
   conn.open "Provider=Microsoft.Jet.OLEDB.3.51;" & _
             "Persist Security Info=False;" & _
             "Data Source=C:\InetPub\WroxArticle2\pim.mdb"
```

As this is a self-calling document (the code in this document is going to call itself, as we will see later on), I need to check *at load time* whether it is performing an update on the user's activity. If the `Request.QueryString("TimeActivity")` collection contains a non-empty string, I can be sure that I am updating the activities. A comma separates the activities of the user, so this:

```
Breakfast,,,,Lunch,,,,,,,,
```

indicates that I have events "Breakfast at 0800 hours" and "Lunch at 1200 hours". I then use the `Split()` function to separate each individual activity, and assign them to an array:

```
   '===Check to see if modification is needed===
   If Request.QueryString("TimeActivity") <> "" Then
      Dim arrTimeActivity
      arrTimeActivity = _
         Split(Request.QueryString("TimeActivity"), ",")
      Counter = 0
```

This array is then used to update the database:

```
      For i = 800 To 2000 Step 100
         strTimeActivity = trim(arrTimeActivity(counter))
         If strTimeActivity <> "" Then
            sql = "INSERT INTO Activity Values (" & dd & _
                  "," & mm & "," & yy & "," & i & _
                  ",'" & strTimeActivity & "')"
            On Error Resume Next
            conn.Execute(sql)
            If Error.Number > 0 Then    '===Record already exists
               sql = "UPDATE Activity Set Activity='" & _
                     strTimeActivity & "' WHERE day=" & dd & _
                     " AND month=" & mm & " AND year=" & _
                     yy & " AND Time=" & i
               conn.Execute(sql)
            End If
```

If the user clears away the activity, I need to delete it from the database:

```
         Else '===See if a deletion is in order
            sql = "DELETE FROM Activity WHERE day=" & dd & _
                  " AND month=" & mm & " AND year=" & yy & _
                  " AND Time=" & i
            conn.Execute(sql)
         End If
         Counter = Counter + 1
      Next
   End If
```

After all the updating (if necessary), I proceed to display all the current activities for the day.

```
'===Show activities===
sqlQuery = "SELECT * FROM Activity WHERE Day=" & dd & _
           " AND Month=" & mm & _
           " AND Year=" & yy & " ORDER BY Time"
set rs = conn.Execute(sqlQuery)
%>

<card id="card1" title="Activity">
   <p>
      Date : <% =dd %>/<% =mm %>/<% =yy %>
      <table columns="2">
         <tr><td>Time</td><td>Activity</td></tr>
      </table>
```

With the snippets above, I'm generating all the different time intervals from 8am to 10pm. The time interval is one hour. I use an `<input>` element to let users modify their activities, and at the same time retrieve the previously saved activity from the database.

Notice that in our case we have thirteen different time intervals, and each `<input>` element has a different name. The first one has the name `TimeActivity800`, the second one is `TimeActivity900`, and so on.

```
      <%
      For i = 800 To 2000 Step 100
         break = False
         While(Not rs.EOF) and Not break
            If i=cInt(rs("Time")) Then
               activity = rs("Activity")
               break = True
            Else
               activity=""
            End If
            If i > cInt(rs("Time")) Then
               rs.MoveNext
            Else
               break = True
            End If
         Wend
      %>

      <% =i %> <input name="TimeActivity<% =i %>"
                           type="text"
                           value="<% =activity %>" />
      <br/>
      <% Activity=""
      Next %>
```

Once the user has keyed in their activities, they can proceed to save the activity into the database:


```
      <do type="accept" label="Update">
         <go href="Activity.asp" method="get">
            <postfield name="dd" value="<% =dd %>" />
            <postfield name="mm" value="<% =mm %>" />
            <postfield name="yy" value="<% =yy %>" />
            <postfield name="TimeActivity"
               value="$(TimeActivity800),$(TimeActivity900),
                      $(TimeActivity1000),$(TimeActivity1100),
                      $(TimeActivity1200),$(TimeActivity1300),
```

```
                        $(TimeActivity1400),$(TimeActivity1500),
                        $(TimeActivity1600),$(TimeActivity1700),
                        $(TimeActivity1800),$(TimeActivity1900),
                        $(TimeActivity2000)" />
            </go>
         </do>
```

A link must be provided to bring the user back to reselect another date:


```
         <do type="option" label="View Calendar">
            <go href="drawDates.asp#card2" method="get">
               <% For j=800 To 2000 Step 100 %>
                  <setvar name="TimeActivity<% =j %>" value="" />
               <% Next %>
            </go>
         </do>
      </p>
   </card>
</wml>
```

That's it! You've seen a dynamic WAP application written using ASP. Note, though, that I haven't been concerned with user authentication, which is something you definitely have to take care if you want to deploy in the real world. To best appreciate the application described in this article, I suggest that you download the source code and try it on your machine!

Developing Dynamic WAP Applications

In the second part of this paper, I shall discuss some of the issues involving in developing dynamic WAP applications:

- Redirecting pages
- Caching on WAP devices
- Cookie support on WAP devices, and maintaining state using the ASP `Session` object
- Maintaining state without cookie support
- Detecting WAP devices
- Passing values from ASP server-side variables to WML client-side variables, and vice versa.
- Tips for using ASP to generate WML content
- Common errors

Redirecting Pages

In web application development, it's not unusual to see a browser being redirected to another web page. This may happen, for example, because you don't have the permissions required to view a page, and you must therefore log in before you can do so.

In ASP, you can use the `Response.Redirect()` method to perform page redirection. Let's take a look at an example.

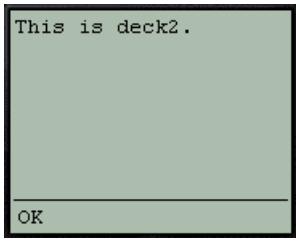
Deck1.asp

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<% Response.Buffer = True %>
<?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="card1deck1" title="Deck1">
      <p>
         This is card 1, but you won't see it...
         <% Response.Redirect "deck2.asp" %>
      </p>
   </card>
</wml>
```

Deck2.asp

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?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="card1deck2" title="Deck2">
      <p>
         This is deck2.
      </p>
   </card>
</wml>
```

When deck1 is viewed using the UP.Simulator, the WAP browser will be redirected to deck2.

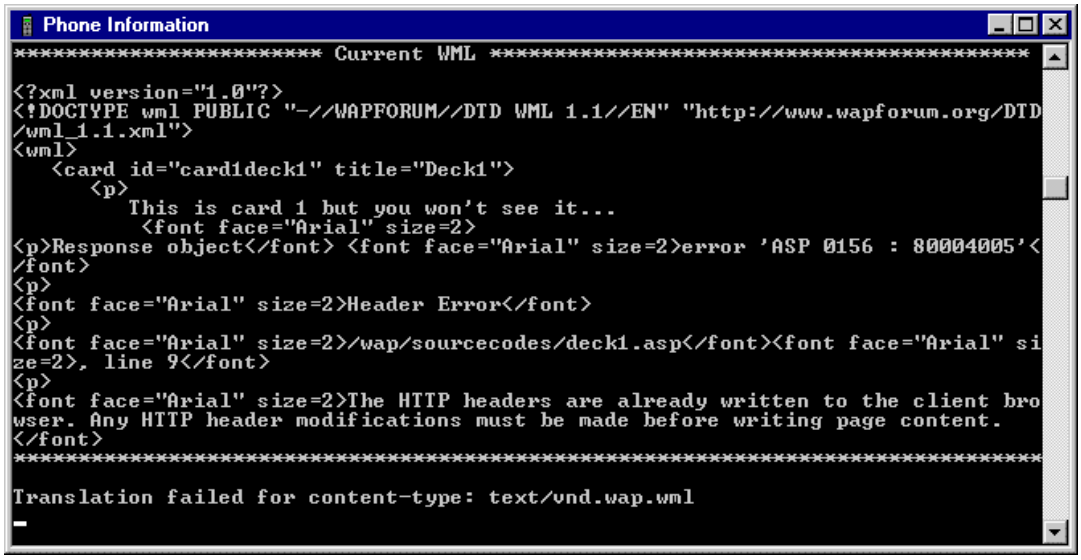


Note that for redirection to occur, you must buffer the page created by the ASP parser using the Response.Buffer property:

```
<% Response.Buffer = True %>
```

Buffering is disabled by default in ASP 2.0 but enabled by default in ASP 3.0.

If a page is not buffered, the web server will send the HTTP headers to the client before the ASP parser has finished parsing the ASP document. In the case of page redirection, this will cause an error. Try setting the `Response.Buffer` property to `False`, and you'll see the error message:



The error message is:

> The HTTP headers are already written to the client browser. Any HTTP header modifications must be made before writing page content.

Caching on WAP Devices

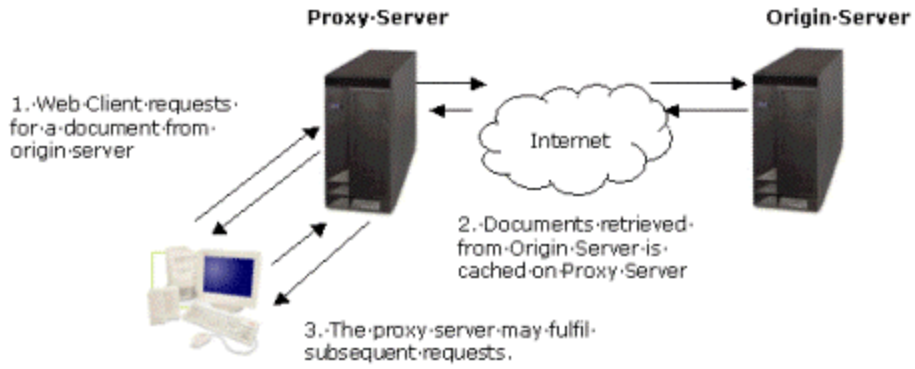
WAP devices generally cache WML decks. This is to reduce the time that is wasted in trying to reload a page that has been loaded earlier. However, for dynamic content, this is not a good feature — the user may be reading some stale information from the cache.

There are two issues to consider when we talk about caching. They are:

- Caching by proxy servers
- Caching by WAP devices

Caching by Proxy Servers

Proxy servers help to reduce the time needed to fetch a document from the origin server. In a network, a proxy server may be set up to act as a halfway house between the Internet and the web clients in the various departments. The proxy server will cache all documents requested by the network, so that it may fulfill subsequent requests for a document retrieved earlier. The trip to the origin server is reduced to making a trip to the proxy server, as the following diagram explains.



Proxy servers should not cache ASP documents, as this defeats the purpose of using ASP in the first place. To turn off caching by proxy servers, use the `Response.CacheControl` property:

```
<% Response.CacheControl="private" %>
```

If for any reason you wish to enable caching by the proxy server, you can set the cache control property to `public`.

```
<% Response.CacheControl="public" %>
```

What about caching by a WAP gateway? The WAP Caching Model specification states that a WAP gateway must *faithfully* implement the role of an HTTP/1.1 proxy with respect to caching and cache header transmission.

Caching by WAP Devices

Just as a proxy server caches pages, WAP devices generally come with some memory to act as a cache. Remember that bandwidth is still a limitation for current WAP devices, so caching WML pages can generally led to improved performance.

While a cache memory helps to load a page that has been loaded previously without making a connection to the origin server, it is a nuisance where time-sensitive pages are concerned. Pages like stock information, weather reports, and online ticketing systems are highly time-sensitive.

By default, WML pages are cached. To force a WML deck to expire immediately after it has been received, I use the `Response.Expires` property:

Expires.asp

```
<%
   Response.ContentType = "text/vnd.wap.wml"
   Response.Expires = -1
%>
<?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="card1" title="NoCache">
      <p>
         <% =now() %>
      </p>
   </card>
</wml>
```

A <meta> element can provide meta-information for a WML deck. Consider the following example:

Meta.asp

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                     "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
   <head>
      <meta http-equiv="Cache-Control"
            content="max-age=0"
            forua="true" />
   </head>
   <card id="card1" title="NoCache">
      <p>
         <% =now() %>
      </p>
   </card>
</wml>
```

The Cache-Control <meta> element allows you to set the caching characteristics of the target WAP device. The content attribute can contain the following values:

- max-age=*time_in_seconds*
- must-revalidate
- no-cache

When the value of content is set to "max-age=0", it expires immediately after it has been loaded. It is equivalent to "no-cache".

We have looked at the two methods for controlling caching on WAP devices, but which is better? In general, controlling the HTTP headers is a better option than using the <meta> element. This is because some WAP gateways may not pass caching-related headers to the WAP device, especially the <meta> elements. Also, <meta> elements are not supported by all WAP devices.

Cookie Support on WAP Devices

At the time of writing, WAP 1.1 devices have very limited support for cookies. If you're planning to make use of ASP's Session object, you have to be aware of this and make appropriate allowances.

The Session Object and Cookies

The ASP Session object requires the use of cookies in order to work correctly. Provided that's the case, using it is straightforward. This is the sample file Session.asp:

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?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="card1" title="Session">
      <p>
         <%
         Session("TimeLogon") = now
         Response.Write "You are logged on at " & _
                        Session("TimeLogon")
         %>
      </p>
   </card>
</wml>
```

Here, this statement,

```
Session("TimeLogon") = now
```

saves the date and time into a `Session` object named `TimeLogon`, and subsequent WML decks can then use this `Session` object to check the logon time of the user. That's all well and good, but since WAP 1.1 devices have limited support for cookies, how can we maintain state *without* using the `Session` object?

Maintaining State without Cookie Support

Let's see now how we can solve the previous problem without using the `Session` object. This is `Nosession.asp`:

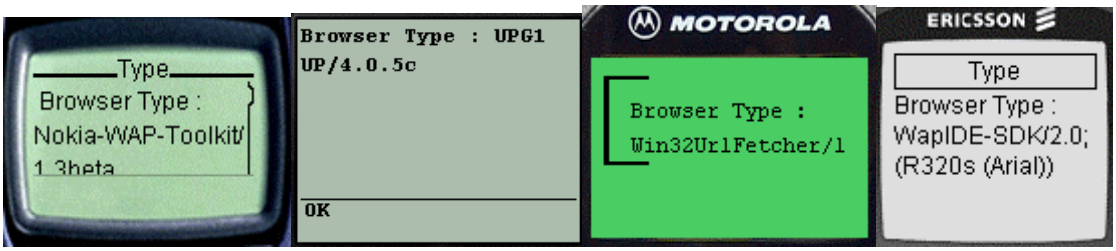
```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                     "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
   <card id="card1" title="Card 1">
      <p>
         <% TimeLogon = now %>
         Welcome!
         <do type="accept" label="Card 2">
            <go href="#card2" method="get">
               <postfield name="TimeLogon"
                          value="<% =TimeLogon %>" />
            </go>
         </do>
      </p>
   </card>

   <card id="card2" title="Card 2">
      <p>
         You have logged on at
         <% =Request.QueryString("TimeLogon") %>
      </p>
   </card>
</wml>
```

Here, I'm passing values from one card to another using the `<postfield>` element. The downside to this using this approach is that it is inconvenient to pass values from one card to another, especially if the number of items to pass increases.

Detecting WAP Devices

To detect what WAP browser the user is using, you can use the `Request.ServerVariables` collection, as I do in `Useragent.asp`:

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?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="card1" title="Type">
      <p>
         Browser Type :
         <% =Request.ServerVariables("HTTP_USER_AGENT") %>
      </p>
   </card>
</wml>
```



It's up to the developer to decide on the level of compatibility that they want to achieve for a particular application. For a list of user agents, point your browser at http://amaro.g-art.nl/useragent/. The following example illustrates how to detect whether a user is using a web browser or a WAP browser to view a page:

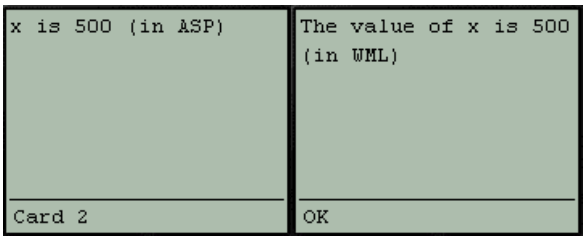
```
<%
   If InStr(Request.ServerVariables("HTTP_USER_AGENT"), _
            "Mozilla") Then
      Response.Redirect "Error.html"
   End If
%>
<% Response.ContentType = "text/vnd.wap.wml" %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                     "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
   <!-- First Card -->
   <card id="card1" title="Welcome">
      <p>
         Welcome...
      </p>
   </card>
</wml>
```

If the user agent contains the word "Mozilla", then the user is using a web browser and I need to redirect them to another HTML page using the `Response.Redirect()` method.

Passing Values from ASP to WML

A common question asked by developers new to WML is how can they pass values between WML client-side variables and ASP server-side variables. The following example illustrates the case:

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?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="card1" title="Card 1">
      <p>
         x is 500 (in ASP)
         <% x = 500 %>
         <do type="accept" label="Card 2">
            <go href="#card2" method="get">
               <setvar name="x" value="<% =x %>" />
            </go>
         </do>
      </p>
   </card>

   <card id="card2" title="Card 2">
      <p>
         The value of x is $(x) (in WML)
      </p>
   </card>
</wml>
```

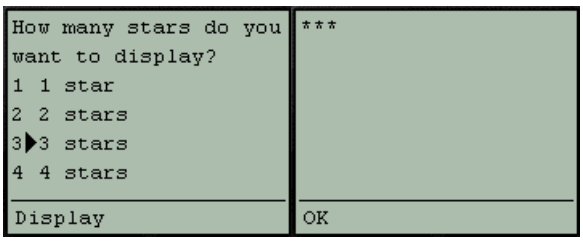


Here, I'm passing the value of x in ASP to the variable in x in WML. I make use of a `<setvar>` element to pass the value from ASP to WML.

Passing Values from WML to ASP

How about the other way round?

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?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="card1" title="Card 1">
      <p>
         How many stars do you want to display?
         <select name="loop">
            <option value="1">1 star</option>
            <option value="2">2 stars</option>
            <option value="3">3 stars</option>
            <option value="4">4 stars</option>
         </select>
         <do type="accept" label="Display">
            <go href="#card2" method="get">
               <postfield name="loop" value="$loop" />
           </go>
         </do>
      </p>
   </card>

   <card id="card2" title="Card 2">
      <p>
         <% loopcount = Request.QueryString("loop")
            For i= 1 To loopcount
               Response.Write "*"
            Next
         %>
      </p>
   </card>
</wml>
```



In this case, I'm using the `<postfield>` element to pass the value to `card2`:

```
<postfield name="loop" value="$loop"/>
```

And in `card2`, I've used the `Request.QueryString` collection to retrieve and assign the value to the variable `loopcount` in ASP:

```
<% loopcount = Request.QueryString("loop")
```

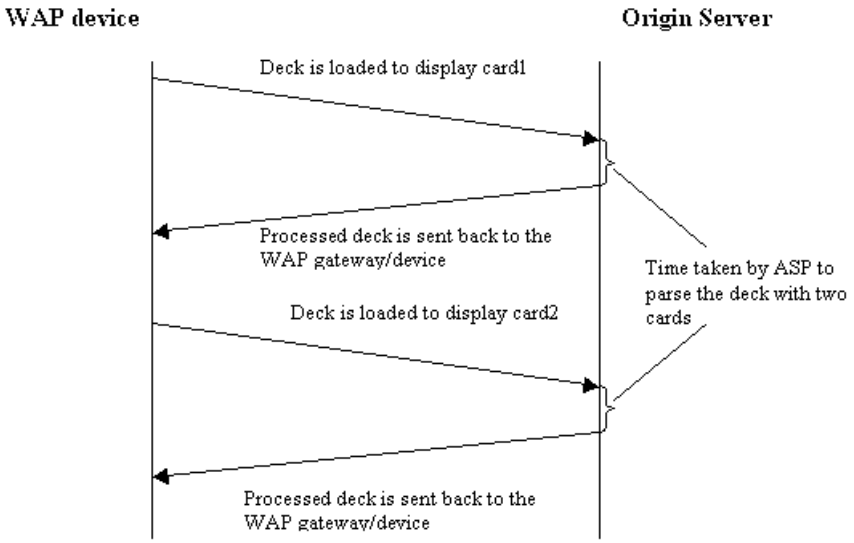
Tips for Using ASP to Generate WML Content

When generating WML content using ASP, bear in mind that a deck can contain multiple cards. If your content for each card is generated dynamically, it may have an impact on the efficiency of your web server. To illustrate this, consider the following example:

```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                     "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
   <!-- First Card -->
   <card id="card1" title="Card1">
      <p>
         Time on card 1 is : <% =time %>
         <do type="accept" label="Card2">
            <go href="#card2" />
         </do>
      </p>
   </card>

   <!-- Second Card -->
   <card id="card2" title="Card2">
      <p>
         Time on card 2 is : <% =time %>
         <do type="accept" label="Card1">
            <go href="#card1" />
         </do>
      </p>
   </card>
</wml>
```

When the ASP parser parses the above deck, both cards will be 'given' the same time. When the user moves from `card1` to `card2`, they will see the same time on both cards. This may not be what we intended — we might want the cards to contain the different times that they were displayed on the WAP device.

To solve this problem, I need to make sure that caching is turned off on the WAP device. For this, I can use the `Response.Expires` property, as described earlier in the paper. However, this causes the deck to be parsed *twice* by the ASP parser, as the WAP device will have to reload the same deck from the origin server to display the second card.



Remember: caching is at *deck level*. When a user moves from a card to another within a deck, using a `Response.Expires` property will not cause a reload of the deck from the origin server.

Hence, when you have dynamic content on multiple cards within a deck, it is important that these cards be separated into different decks. The diagram showing the request and response between the WAP device and origin server is valid *provided* caching can be done at the *card* level.

As can be seen, every time the WAP device requests the same deck, the ASP server spends time parsing it. If the deck contained a lot of cards, the time needed would be substantial: this is inefficient. Instead, by breaking the cards down into multiple decks, the time spent parsing individual decks is reduced.

Common Errors

Finally, let's briefly examine some of the common errors that developers may encounter when using ASP to generate WML content.

Not setting the correct MIME type
The WML MIME type must be set using the `Response.ContentType` property, like this:

```
<% Response.ContentType = "text/vnd.wap.wml" %>
```

Cookies are not supported
When using cookies or the `Session` object, always ensure that the target platform is capable of supporting cookies.

The `POST` method may not work correctly on some WAP devices
When using a `<postfield>` element, be aware that some WAP devices (the Nokia 7110, for instance) have problems in supporting the `POST` method.

You need a web server to run ASP!
Very often, people just run ASP documents straight from their local hard disk. ASP documents require a web server (in particular, the ASP parser) to process them, so that the resulting page can be sent to the client. Also important point to note is that you need to use the http:// syntax, and *not* the physical filename.

Summary

In this paper, I have taken a look at how to create a dynamic WAP application using ASP. I have also highlighted some of the important points to take note of when developing dynamic WAP applications. As WAP applications tend to be highly time-sensitive, the ability to create dynamic applications is an important asset to the developer.