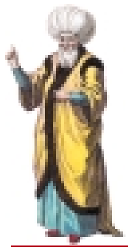


CHAPTER 19

Using ASP to generate dynamic WAP content

19.1 INTRODUCTION

In chapter 17, “Introduction to Microsoft Active Server Pages,” you learned how to use ASP to create dynamic web applications, and in chapter 18, “Database Connectivity with ADO,” you learned how to simplify back-end database access using some of the current database technologies, ADO and OLE DB.

In this chapter, you will use your new skills to create a dynamic WAP application. This chapter will also compare the development processes used to create web and WAP applications.

19.2 CREATING A DYNAMIC WAP APPLICATION

Let’s now develop our first WAP application using ASP.

```
Firstwap.asp      ❶
<% Response.ContentType = "text/vnd.wap.wml" %>   ❷
<?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>
     It is now <% =time %> and this is my first dynamic WAP application!   ❸
   </p>
 </card>
</wml>
```

Code comments

❶ Saves our file with a .asp extension. Since this is a dynamic WAP application, we create an ASP document so that the web server can process the scripts in the document.

❷ All .wml files are associated with the WML MIME type (text/vnd.wap.wml) so that the WAP browser can display.

❸ Uses the `time()` function to return the current time.

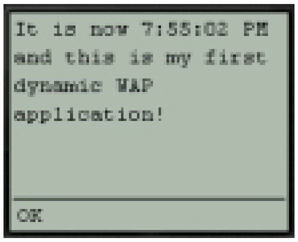
This is accomplished by the `Response.ContentType` property:

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

The only thing that is truly dynamic in this application is the line:

```
It is now <% =time %> and this is my first dynamic WAP application!
```

Here, we are calling the `time()` function in VBScript, then inserting the value into the WML <p> element.


It is now 7:55:02 PM and this is my first dynamic WAP application!

OK

Figure 19.1 Dynamic WAP application

Where is the development environment for developing WAP applications? If you are a Visual Basic or Visual C++ developer, you are accustomed to seeing the helpful integrated development environment (IDE). There aren’t many development environments to use for developing WAP applications (though the Nokia WAP Toolkit and the Ericsson WAP IDE provide something similar to an IDE). Our recommendation is to use your favorite text editor to key in the codes.

Depending on the time you run this application, you will see something like figure 19.1.

19.3 TESTING USING WAP EMULATORS

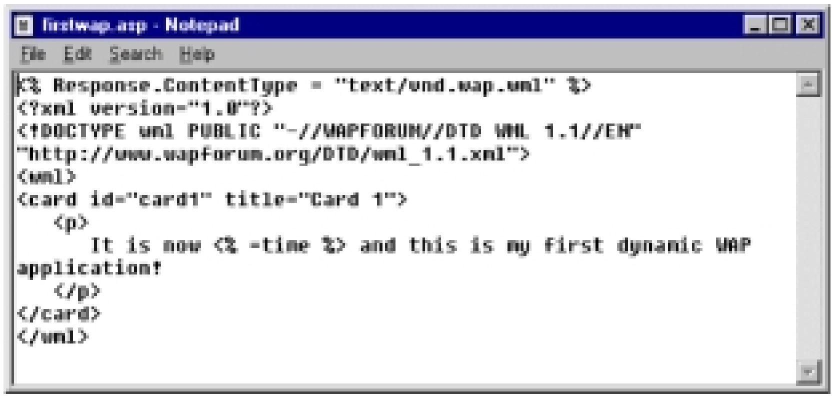
How to use WAP emulators is the most frequently asked question on many of the WML mailing lists. Because we have received numerous emails asking how to run a WAP application using one or more emulators, we are going to go through the process of creating a dynamic application, from editing to success—where you can sit back and enjoy watching your application as it runs beautifully in the emulator. Chapter 2, "WAP Application Development Platforms," covers some of the emulators that you can use for your development purposes.

You'll need to complete four steps before you get to the success stage:

- Create and edit an ASP document
- Save the ASP document to the appropriate directory
- Install and run the emulator
- View your application from the perspective of multiple emulated WAP devices

19.3.1 Step 1: editing an ASP document

You may be accustomed to your favorite ASP development tool, such as Microsoft Visual Interdev 6.0. But my favorite is still Notepad (figure 19.2). In any case, edit your ASP document in a text editor (any text editor capable of saving your file in plain text format will do; do not use a word processor such as Microsoft Word to type in your codes) and save it to your web-publishing directory (see step 2).



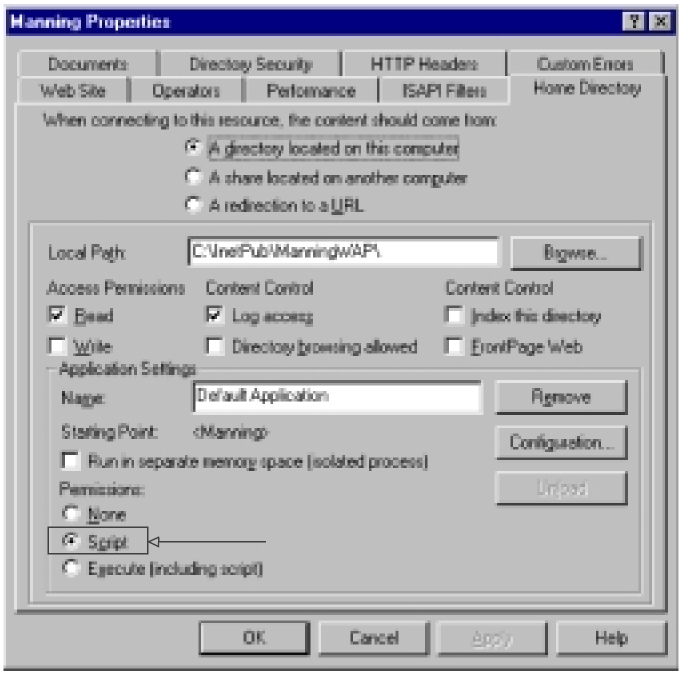
```
<% Response.ContentType = "text/vnd.wap.wml" %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="card1" title="Card 1">
   <p>
      It is now <% =time %> and this is my first dynamic WAP
application!
   </p>
</card>
</wml>
```

Figure 19.2 Editing a WAP application in Notepad

19.3.2 Step 2: saving to the web-publishing directory

To execute ASP scripts, you need a web server (see appendix D). For my development use, I like Microsoft Internet Information Server 4.0. My default web publishing (home) directory is c:\inetpub\manningWap\. For the examples in this chapter, I will save all my .wml and .asp documents in my home directory. You may configure your home directory to some other physical directory. In this case do save your files in that particular directory.

Be sure that you have selected the Script permissions (figure 19.3) for this directory; otherwise, the web server will not be able to process the ASP document.



When connecting to this resource, the content should come from:
[✓ A directory located on this computer]
A share located on another computer
A redirection to a URL

Local Path: C:\InetPub\Manning\wAP\

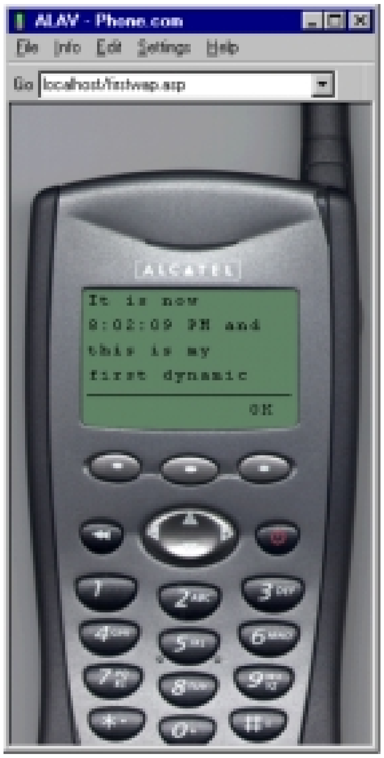
Access Permissions: [✓ Read] Write
Content Control: [✓ Log access] Directory browsing allowed
Content Control: Index this directory, FrontPage Web

Application Settings
Name: Default Application
Starting Point: <Manning>
Run in separate memory space (isolated process)
Permissions: None [✓ Script] Execute (including script)

Figure 19.3
Selecting the Script permissions

If you are not sure how to set this permission, refer to your web server documentation.

19.3.3 Step 3: using the emulators



It is now
9:02:09 PM and
this is my
first dynamic
OK

Figure 19.4 Using an emulator

Once your ASP document is saved to the proper directory, test it using the emulators. There are quite a number of emulators available; for this section, I am going to use Phone.com's UP.Simulator to test our application.

After installing the emulator, launch it and open your ASP document. You should see something like figure 19.4.

To view a WML deck or ASP document in the emulator, enter the URL in the Go textbox (figure 19.5).

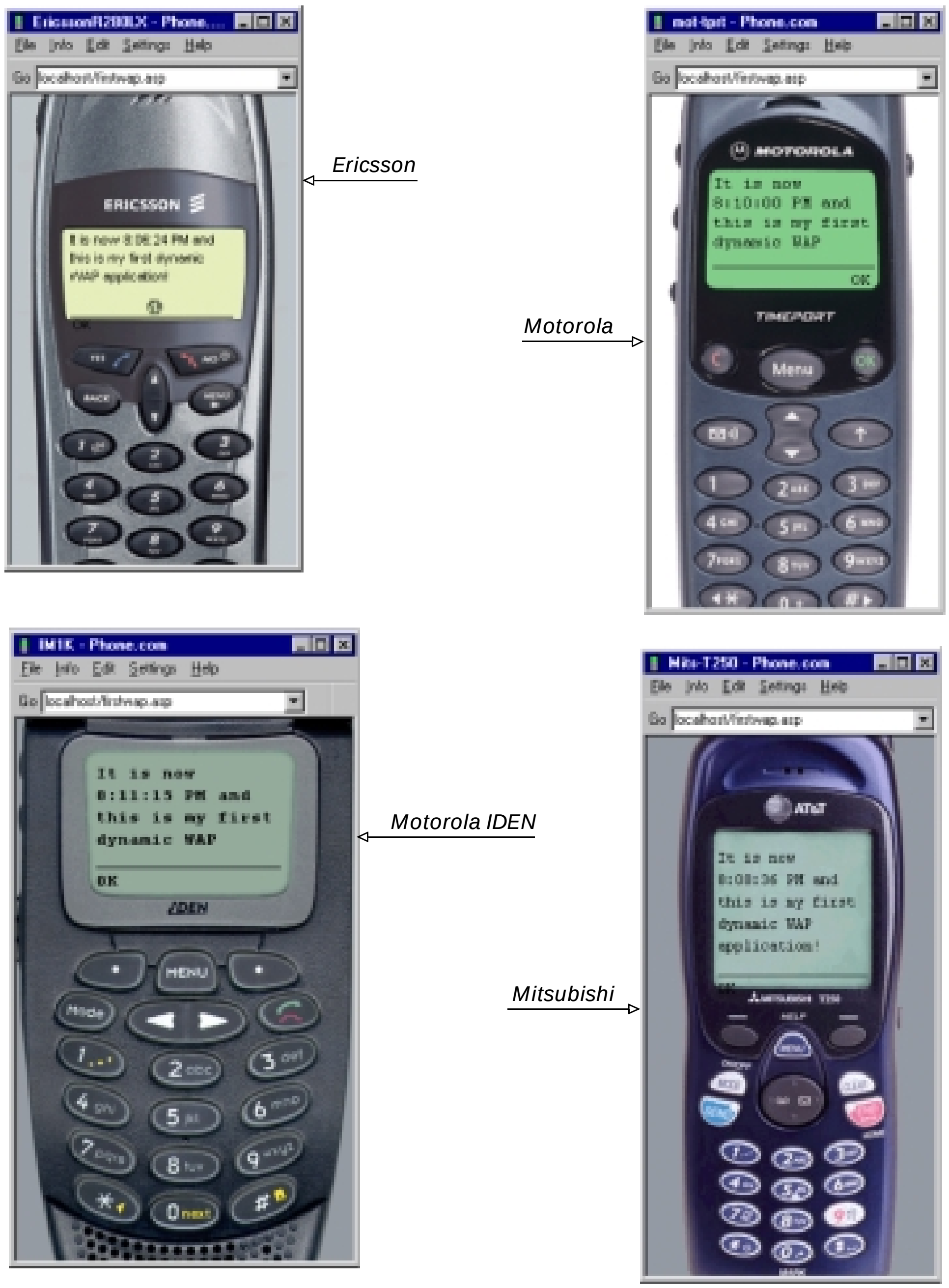
Figure 19.5 The Go textbox

It's so easy, isn't it? You are on your way to professional WAP development.

19.3.4 Step 4: testing the look and feel

Although this step is optional, I strongly suggest performing it if you are developing applications to be deployed on a variety of devices. One of the nice things about emulators is that they allow you to choose from numerous emulated devices for testing. At the moment, Phone.com provides different browser "skins" to emulate the behavior of different devices. It is good practice to run your application on different devices to see how the look and feel differs.

Figure 19.6 shows how our application looks in various emulated devices:



It is now 8:06:24 PM and this is my first dynamic WAP application!

It is now 8:10:00 PM and this is my first dynamic WAP

It is now 8:11:15 PM and this is my first dynamic WAP

It is now 8:08:36 PM and this is my first dynamic WAP application!

Figure 19.6 Same application, different emulators

You might not notice much difference among the devices for this application. However, if you have a much more complex application, the differences become very noticeable.

Other emulators that you can try are:

- Nokia WAP Toolkit 2.0
- Ericsson WAP IDE 2.1
- Motorola Mobile ADK (MADK)

For more information about downloading and installing emulators, see chapter 2.

19.4 SENDING AND RETRIEVING DATA

A useful WAP application requires participation and input from the user. You may need the user to supply his logon password, or you may need the user to key in a quantity for an item that he is buying using a WAP handphone. In this case, your WAP application must be able to retrieve the data sent from the WAP browser and process it.

19.4.1 Passing values from client to server

Chapter 5 explained entering information using the WML `<input>` element. However, information that is entered must be sent to the web server for processing before the application can be considered useful.

This WML code allows the user to enter his loginID and password:

```
Login.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>
    LoginID: <input type="text" name="Login" maxlength="8"/>
    Password: <input type="password" name="Password" maxlength="8"/>
    <do type="accept" label="Login!">                        ❶
       <go method="post" href="Authenticate.asp">            ❷
          <postfield name="Login" value="$Login" />
          <postfield name="Password" value="$Password" />   ❸
       </go>
    </do>
  </p>
</card>
</wml>
```

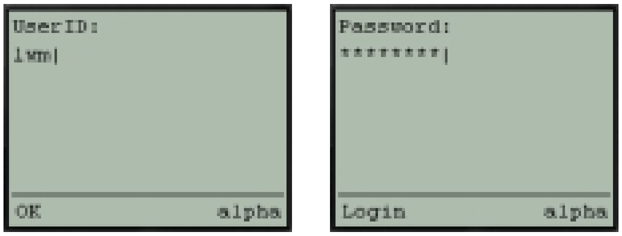
Uses the `<input>` elements for text input

Code comments

❶ Maps a function to a soft key using the `<do>` element.

❷ Indicates that the information be sent to the server using the `POST` method.

❸ Once the loginID and password are entered, use the WML `<postfield>` element to send the information to the ASP document named Authenticate.asp (figure 19.7).



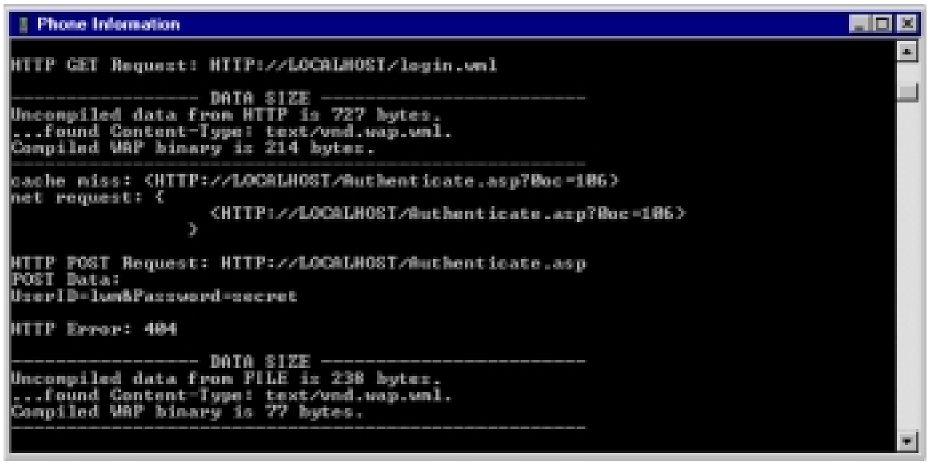
```
UserID:
lwm|
OK                alpha
```

```
Password:
********|
Login             alpha
```

Figure 19.7 Entering the loginID and password

19.4.2 Using the POST method

The `POST` method sends data to the server in a separate transmission. The client first contacts the document listed in the `href` attribute of the WML `<go>` element, in this case, `authenticate.asp`, and then it sends the data across. To see what goes on behind the scenes when this transmission occurs, you can use the UP.Simulator to test the application and examine the results in the Phone Information window (figure 19.8).



```
HTTP GET Request: HTTP://LOCALHOST/login.wml

------------------- DATA SIZE -------------------------
Uncompiled data from HTTP is 727 bytes.
...found Content-Type: text/vnd.wap.wml.
Compiled WAP binary is 214 bytes.
-------------------------------------------------------
cache miss: <HTTP://LOCALHOST/Authenticate.asp?Boc=186>
net request: {
                  <HTTP://LOCALHOST/Authenticate.asp?Boc=186>
                }

HTTP POST Request: HTTP://LOCALHOST/Authenticate.asp
POST Data:
UserID=lwm&Password=secret

HTTP Error: 404

------------------- DATA SIZE -------------------------
Uncompiled data from FILE is 238 bytes.
...found Content-Type: text/vnd.wap.wml.
Compiled WAP binary is 77 bytes.
-------------------------------------------------------
```

From the Phone Information window, you can see that the client first requests the document Authenticate.asp and then sends the data, `Login=lwm&Password=secret`, in a separate transmission.

19.4.3 Using the GET method

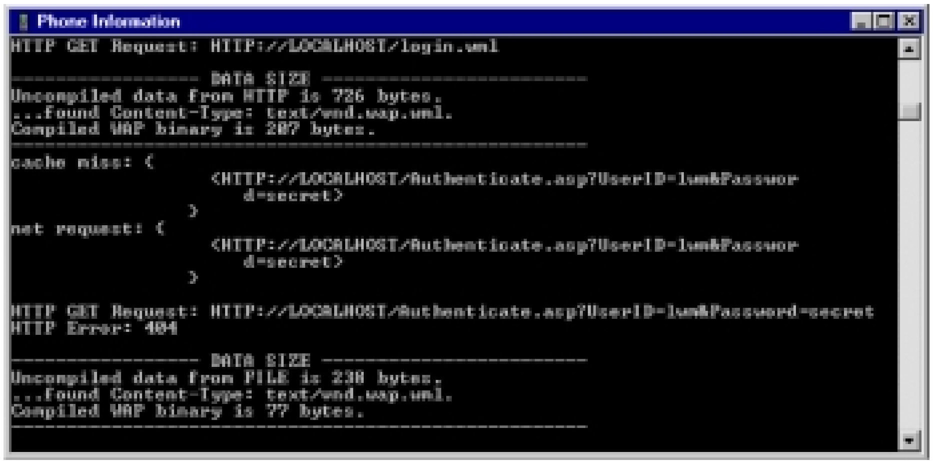
Let's modify this application so that it uses the `GET` method to send data:

```
<go method="get" href="Authenticate.asp">
```

Notice that this time (figure 19.9), the data to be sent is appended to the URL:

```
HTTP GET Request:
HTTP://LOCALHOST/Authenticate.asp?Login=lwm&Password=secret
```

ERROR 404 Do not worry about the HTTP Error: 404 message in figure 19.9. This error occurs because the Authenticate.asp document could not be found in the web-publishing directory.



```
HTTP GET Request: HTTP://LOCALHOST/login.wml

------------------- DATA SIZE ---------------------------
Uncompiled data from HTTP is 726 bytes.
...found Content-Type: text/vnd.wap.wml.
Compiled WAP binary is 207 bytes.
---------------------------------------------------------
cache miss: {
                  <HTTP://LOCALHOST/Authenticate.asp?UserID=lwm&Passwor
                     d=secret>
                }
net request: {
                  <HTTP://LOCALHOST/Authenticate.asp?UserID=lwm&Passwor
                     d=secret>
                }

HTTP GET Request: HTTP://LOCALHOST/Authenticate.asp?UserID=lwm&Password=secret
HTTP Error: 404

------------------- DATA SIZE ---------------------------
Uncompiled data from FILE is 238 bytes.
...found Content-Type: text/vnd.wap.wml.
Compiled WAP binary is 77 bytes.
---------------------------------------------------------
```

Figure 19.9 The Phone Information window using the POST method

19.4.4 A common pitfall using the GET method

HTML programmers who are familiar with the `GET` method often neglect to encode special characters as in:

```
Action="Authenticate.asp?Name=lwm&Password=secret"
```

You might be tempted to do the same in WML:

```
<go href="Authenticate.asp?Name=lwm&Password=secret">
```

If you do, your code will fail. In WAP, the ampersand (&) must be encoded with the special code: `&amp;`

So your code must be modified as follows:

```
<go href="Authenticate.asp?Name=lwm&amp;Password=secret">
```

19.5 Retrieving values sent to the server

Depending on the method you use to send data to the server, you can either use the Request.Form or the Request.QueryString collection to retrieve the values from the server.

19.5.1 Using the Request.Form collection

If you use the POST method to send data, use the Request.Form collection to retrieve the values:

```
<%
Dim Name, Password
Name =Request.Form("Name")
Password = Request.Form("Password")
...
%>
```

If you use the GET method, use the Request.QueryString collection:

```
<%
Dim Name, Password
Name =Request.QueryString("Name")
Password = Request.QueryString ("Password")
...
%>
```

POST ERROR Before you deploy your application, test it on a real device to ensure that the handset supports the send method that you are using.

19.6 SESSION SUPPORT IN WAP DEVICES

If you are an ASP web developer, you are familiar with the Session object discussed in chapter 17. The Session object allows the web server to maintain state between itself and the client. As we also saw in chapter 17, the Session object requires cookie support.

Unfortunately, the current generation of WAP phones does not support cookies. Fortunately, there are simple ways to test whether a WAP device supports cookies.

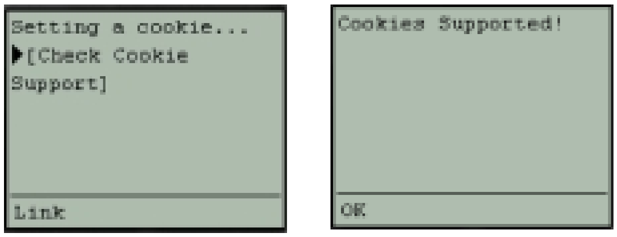
The first application attempts to set a cookie on the WAP device. The second then attempts to retrieve the cookie value.

```
<% Response.ContentType = "text/vnd.wap.wml"
   Response.Buffer = true              <- Sets the web server buffering to 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="card1" title="Cookie">
     <p>
       Setting a cookie...
       <% Response.Cookies("Test")= "123" %>    <- Sets a cookie using the Cookies collection of the Response object
       <a href="getCookieValue.asp">Check Cookie Support</a>    <- Sets an anchor to link to another ASP document
     </p>
  </card>
</wml>
```

If the cookie value can be retrieved, the WAP device supports cookies (figure 19.10).

```
<% 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="Cookie">
      <p>
         <%
             if Request.Cookies("Test") = "123" then
                 Response.Write "Cookies Supported!"
             Else
                 Response.Write "No Cookie-Support!"
             end if
         %>
      </p>
   </card>
</wml>
```

If the Cookies collection "Test" contains the value "123", cookie is supported, else cookie not supported



```
Setting a cookie...
▶[Check Cookie
Support]
Link
```

```
Cookies Supported!
OK
```

Figure 19.10
Checking for cookie support

WAP GATEWAYS Before you lament the lack of cookie support on current WAP devices, fret not! Some of the WAP gateways actually do the job of supporting cookies for the devices. The WAPlite gateway, discussed in more detail in chapter 27, supports session and persistent cookies. The rule-of-thumb for developing WAP applications at the moment is to forget about cookies!

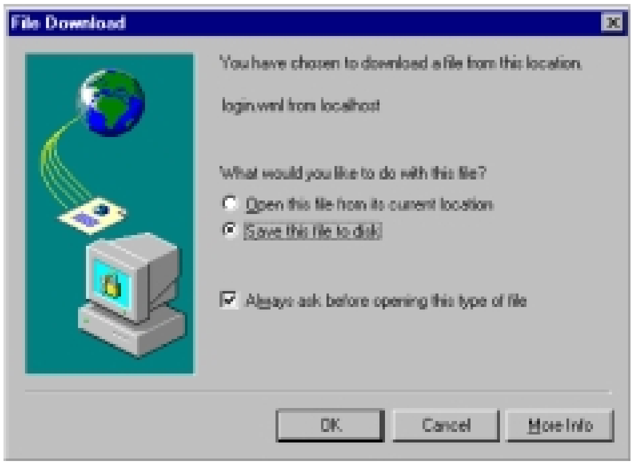
Emulators often support cookies, so don't be fooled if your application using the Session object seems to work well on an emulator. Test it on a real device!

Once you have verified cookies support, you can use the Session object as discussed in chapter 17.

19.7 USING ENVIRONMENT VARIABLES

When developing applications for your services, it is essential that you make an attempt to detect the correct browser type. For example, if you try to access a WAP application from a web browser, you may see something like figure 19.11.

It would be far better if your application could first detect that the user is using a web browser, then redirect him to a web page that explains the error. Using specific environment variables, you can add this functionality to your WAP application.


File Download

You have chosen to download a file from this location.

login.wml from localhost

What would you like to do with this file?

( ) Open this file from its current location
[✓ Save this file to disk]

[✓] Always ask before opening this type of file

Figure 19.11
Accessing a WAP application from a web browser

19.8 Detecting web and WAP browsers

A very simple detection method is to check the value of either `HTTP_USER_AGENT` or `HTTP_ACCEPT`.

Getting the value of HTTP_USER_AGENT

This code detects whether the user is using a WAP or a web browser based on the value of the `HTTP_USER_AGENT` environment variable.

```
<% 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="card1" title="Detecting...">
   <p>
       <% if Instr(Request.ServerVariables("HTTP_USER_AGENT"), "Moz") then
             Response.Redirect "nonWML.html"
          Else
             Response.Write "Welcome to the Wireless World!"
          end if
       %>
   </p>
</card>
</wml>
```

If the `HTTP_USER_AGENT` variable contains the word "Moz", redirect him to nonWML.html, else write a welcome message

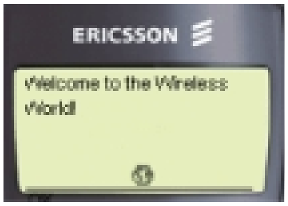

Welcome to the Wireless World!

Figure 19.12 Using a WAP browser

If the user is using a WAP browser (in this case, an emulator), a card like the one illustrated in figure 19.12 appears.

If the user is using a web browser, he will be redirected to another page as illustrated in figure 19.13:

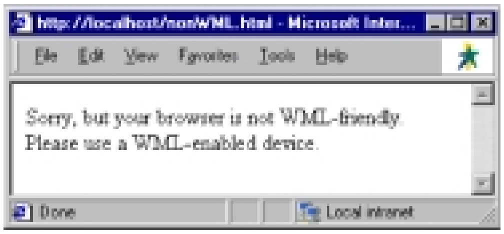

Sorry, but your browser is not WML-friendly.
Please use a WML-enabled device.

Figure 19.13 Redirecting the HTML user

Getting the value of HTTP_ACCEPT

Another method of differentiating a WAP browser from a web browser is to use the environment variable HTTP_ACCEPT.

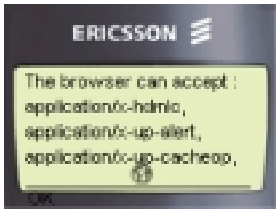
This is the call via Detect.asp:

```
<% 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="card1" title="Detecting...">
   <p>
       <% if Instr(Request.ServerVariables("HTTP_USER_AGENT"), "Moz") then
             Response.Redirect "HTMLaccept.asp"
          else
             Response.Write "Welcome to the Wireless World!"
             Response.Write "The browser can accept : " &
             Request.ServerVariables("HTTP_ACCEPT")
          end if
       %>
   </p>
</card>
</wml>
```

This is the call via HTMLaccept.asp:

```
<HTML>
<TITLE>HTML Accept</TITLE>
<BODY>
```

```
<% Response.Write "The browser can accept : " &
   Request.ServerVariables("HTTP_ACCEPT") %>
</BODY>
</HTML>
```



ERICSSON

The browser can accept :
application/x-hdmlc,
application/x-up-alert,
application/x-up-cacheop,

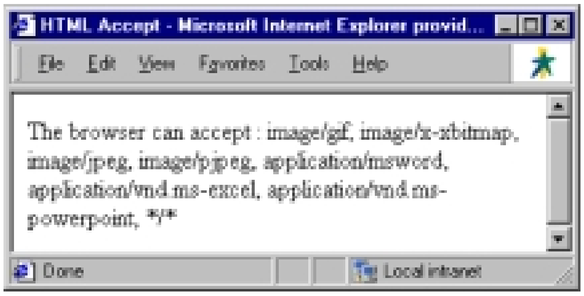
Figure 19.14 The Phone.com browser

The Phone.com WAP browser accepts documents of the MIME types in table 19.1. (Note that Ericsson phones in the U.S. use the Phone.com's browser shown in figure 19.14.)

Table 19.1 Phone.com WAP browser MIME types

MIME type	Description
application/x-hdmlc	HDML compiled
application/x-up-alert	Alert for the Phone.com browser (sent to phone's alert inbox)
application/x-up-cacheop	Clears a Phone.com browser's cache
application/x-up-device	Defines the handset requesting information
application/x-up-digestentry	Obsolete, unsupported type
application/vnd.wap.wml	WML compiled
text/x-wap.wml	WML decks
text/vnd.wap.wml	WML decks
application/vnd.wap.wmlscript	WMLScript compiled
text/vnd.wap.wmlscript	WMLScript uncompiled
application/vnd.uplanet.channel	Obsolete, unsupported type
application/vnd.uplanet.list	Obsolete, unsupported type
text/x-hdml	HDML files uncompiled
text/plain	ASCII text
text/html	HTML files
image/vnd.wap.wbmp	Image in WBMP format
image/bmp	Image in BMP format (dependent on WAP devices, not supported by all)
application/remote-printing text/x-hdml;version=3.1	HDML format version 3.1
text/x-hdml;version=3.0	HDML format version 3.0
text/x-hdml;version=2.0	HDML format version 2.0

The IE5 browser (figure 19.15) accepts the MIME types listed in table 19.2.


HTML Accept - Microsoft Internet Explorer provid...
File Edit View Favorites Tools Help
Done
Local intranet

The browser can accept : image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword, application/vnd.ms-excel, application/vnd.ms-powerpoint, */*

Figure 19.15
The IE5 browser

Table 19.2 IE5 browser MIME types

MIME type	Description
Image/gif	Image in GIF format
Image/x-xbitmap	Image in Xbitmap format
Image/jpeg	Image in JPEG format
Image/pjpeg	Image in PJPEG format
Application/msword	Microsoft Word Application
Application/vnd.ms-excel	Microsoft Excel Application
Application/vnd.ms-powerpoint	Microsoft PowerPoint Application
/	All types accepted

Looking for the string `"text/vnd.wap.wml"` in the `HTTP_ACCEPT` string can help you determine if your user is using a WAP browser.

The `HTTP_ACCEPT` string also indicates whether a WAP device provides image support (image/vnd.wap.wbmp).

In general, it is safer to check the `HTTP_ACCEPT` string as the Microsoft Mobile Explorer is reputed to return a string containing the word “Mozilla” in the `HTTP_USER_AGENT` string!

19.9 DETECTING WAP DEVICES

While identifying a WAP browser from a web browser is useful, the real challenge for the WAP developer is detecting the various makes of handsets and devices used to access your WML application.

While WML is a specification defined by the WAP forum, every WAP device renders WML differently. This can produce a different look and feel in your application. As we noted in chapter 15, an application that is designed to work on one device may behave differently on another. To make things worse, different devices have different memory constraints (we will discuss this in a later section).

So how do we effectively identify the WAP browser in use?

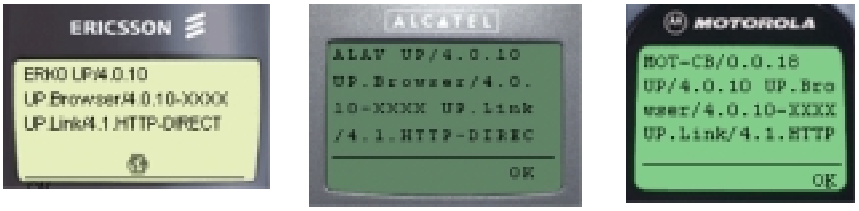
Fortunately, every WAP device has a unique identifier string and this string can be retrieved using the environment variable HTTP_USER_AGENT.

This application returns the user agent string of the device being used:

```
<% 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="User Agent">
  <p>
     <% = Request.ServerVariables("HTTP_USER_AGENT") %>
  </p>
</card>
</wml>
```

Uses the ServerVariables collection of the Request object to access the value of HTTP_USER_AGENT

Figure 19.16 shows the user agent string for three devices:



```
ERK0 UP/4.0.10
UP.Browser/4.0.10-XXXX
UP.Link/4.1.HTTP-DIRECT
```

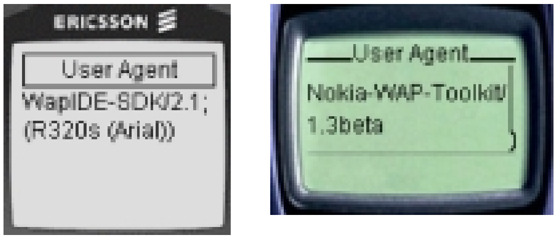
```
ALAV UP/4.0.10
UP.Browser/4.0.
10-XXXX UP.Link
/4.1.HTTP-DIREC
                OK
```

```
MOT-CB/0.0.18
UP/4.0.10 UP.Bro
wser/4.0.10-XXXX
UP.Link/4.1.HTTP
                OK
```

Figure 19.16 User agent string on three emulations

The emulated devices illustrated utilize Phone.com's WAP browser. Note that each device has a unique user agent string. Remember that certain phones may utilize different browsers in different countries (the Ericsson phone in figure 19.16 is using Phone.com's browser).

Figure 19.17 shows emulators from Ericsson and Nokia. They are currently showing the user agent string:



```
User Agent
WapIDE-SDK/2.1;
(R320s (Arial))
```

```
User Agent
Nokia-WAP-Toolkit/
1.3beta
```

Figure 19.17 Emulators showing the user agent string

Identifying the browser is simply a matter of looking for the keyword in the HTTP_USER_AGENT string:

```
UAString = Request.ServerVariables("HTTP_USER_AGENT")
if Instr( UAString, "Nokia") then
```

Keyword is "Nokia" for Nokia devices

```
   Response.Redirect "/Nokia/index.wml"
elseif Instr( UAString, "UP") then
   Response.Redirect "/UP/index.wml"
else
```

Keyword is "UP" for devices using Phone.com's browser

Before you deploy your application, it is imperative to check the exact user agent strings for the various devices.

19.10 TESTING USING A REAL HANDSET

While emulators provide a realistic feel of how users will experience your application, nothing beats the real device.

Once you have developed your application, the best thing to do next is test your application on as many emulators as possible. This is important if your target audience is a large group that uses different devices. The vendor-provided emulators are essential for estimating the look and feel of your application in an actual device.

Once you have tested your application using one or more emulators, the next level of testing should involve devices that will most likely be used to run your application. If you are creating an in-house application (e.g., only for internal staff use), it is less tricky since the WAP device used can be determined by company policies. If you are developing a service for a broad group of users, it is worthwhile to get as many WAP devices as possible for testing purposes.

Testing on a real handset can uncover numerous problems:

Problems	Description
Caching	If your application explicitly disables caching on the WAP device, it is important to test this functionality on a real handset, as emulators may not function correctly.
Cookie support	The most notorious culprits in breaking your application are cookies or sessions. Most emulators support cookies, but many actual handsets do not. Remember that when testing for cookie support, the WAP gateway plays a part. For more information about WAP gateways, see chapter 27.
GET and POST methods	Emulators have no problem sending your data to the web server using either the `GET` or `POST` method. But when it comes to real handsets, some devices may not function correctly. You will only know if there is a problem by testing your application on a real handset.
Look and feel	For platforms that do not have an emulator such as the Siemens phones (though it uses the Phone.com's browser), testing on the real device is the only way to ascertain the look and feel of your application.
Maximum size of WAP binary	Testing your application on a real handset may help you to uncover the maximum size limit of the WAP device.
Usability	The most overlooked aspect of creating WAP applications. On the emulator, it is easy to enter characters into the phone. Try that on a real handset and you will appreciate this point!

19.11 SIZE CONSTRAINTS OF WAP DEVICES

As you recall, the basic unit of information transferred from the origin server to the WAP device is known as a deck. To minimize the amount of data sent to the WAP device, a deck is compiled into bytecode format known as WAP binary.

Very often, beginning WAP developers tend to overlook the limitations of WAP devices. Because WAP devices have limited memory, the WAP binary that is sent to the device must not exceed its memory capacity. Different devices have different memory constraints. Table 19.3 describes the limitations of some popular WAP devices.

Table 19.3 Limitations of WAP devices

WAP browser	Maximum WAP binary size
UP.Browser 3.2	1492 bytes
UP.Browser 4.x	2048 bytes
Ericsson R320	Approximately 3000 bytes
Ericsson R380	Approximately 3500 bytes
Ericsson MC218	More than 8000 bytes
Nokia 7110	1397 bytes

Failure to adhere to the memory constraint imposed by the device will cause the deck to be incorrectly loaded on the device.

In general, it is useful for developers to detect the kind of devices that the user is using and, based on that information, send different versions of a deck to suit the limitations of the devices.

19.12 CONTROLLING PAGE CACHING

When you load a web page using a web browser, the page is saved to your hard disk so that the next time you request the same page, it is accessed from your hard disk, instead of from the origin server. This is known as *browser caching*.

There is another caching method known as *proxy caching*. Using this method, a dedicated server acts as a go-between from the web surfer to the origin server. The dedicated server is known as a *proxy server*.

An example of proxy caching is when a proxy server caches web pages for an organization. When a user requests a page, it is saved to the proxy server's hard disk. When another user from the same organization requests the same page, the proxy server satisfies the request (figure 19.18).

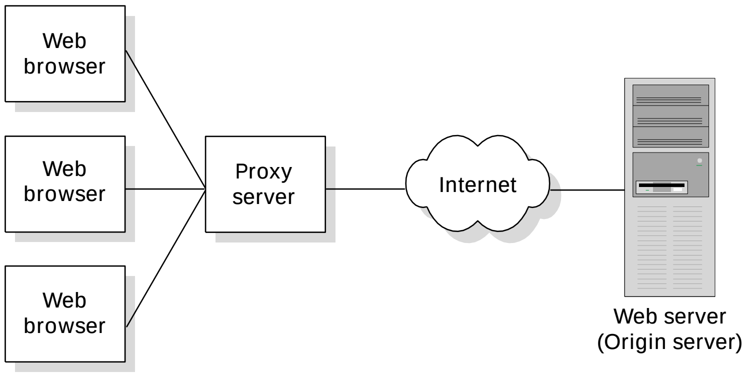


Figure 19.18 Caching at the web browser level and at the proxy server level

As you remember from chapter 1, a WAP device communicates with a WAP gateway. Thus in the WAP caching model, there are three levels of caching (figure 19.19):

- WAP device caching
- WAP gateway caching
- Proxy server caching

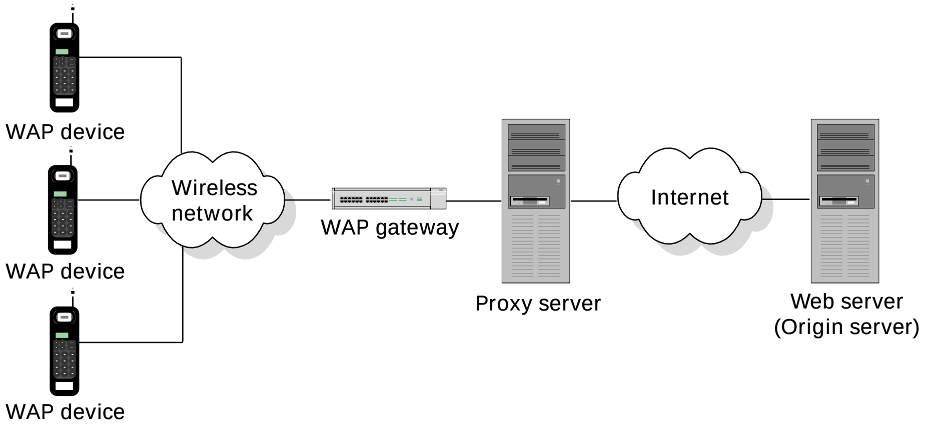


Figure 19.19 Caching at the WAP device level, the WAP gateway level, and proxy server level

The motivation for caching is to reduce the time required to load a document from the origin server. Once a document has been requested and cached, it can be reused. However, if you cache time-sensitive pages, such as stock quotes and weather reports, you are defeating the purpose of caching.

One notable difference between the web model and the WAP model is that WAP is commonly used for dynamic information retrieval. WAP is never intended to replace the web browser as a device for "surfing" the web. Due to the limited size and capability of the WAP device, it is most commonly used to retrieve dynamic data, such as stock information. As such, the information content is highly volatile and time-sensitive. Caching WAP pages in this case makes no sense.

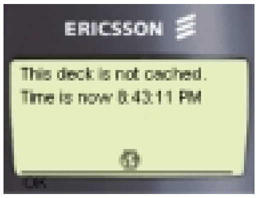
While caching WAP pages has its disadvantages, there are merits. For example, given the limited bandwidth of a WAP device, it would be good to cache frequently accessed static pages. Such pages might include the welcome screen of a site, the login screen for a secure site, and other nonchanging screens.

19.12.1 Disabling caching

To disable caching on a WAP device, use the ASP `Response.Expires` property. This WML deck is not cached when it is loaded:

```
<% 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>
       This deck is not cached.
       Time is now <% =time %>
   </p>
</card>
</wml>
```

Causes the deck to expire using the `Expires` property of the Response object



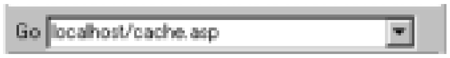
This deck is not cached.
Time is now 8:43:11 PM

Figure 19.20 Caching is disabled

Figure 19.20 shows how the application appears using an emulated WAP device.

To convince yourself that the deck is not cached, load it in your WAP emulator and load the same page again (do not use the reload/refresh function in your emulator).

PECULIARITY On the Phone.com browser, if you load the deck and then click the Return key in the Go textbox again, the same deck is loaded from the cache.



localhost/cache.asp

Figure 19.21 Loading cache.asp

To work around this problem, load the deck in figure 19.21 (cache.asp) and then load another independent deck. After that, load the first deck again. This time around, the first deck is fetched again from the origin server.

The `Response.Expires` property sets the time when the deck expires in the WAP device's cache. If you want the deck to expire five minutes after it has been loaded, you can set:

```
Response.Expires = 5
```

A commonly used method is to set

```
Response.Expires = 0
```

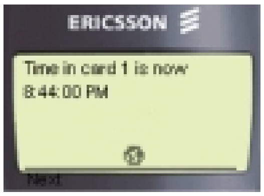
However, due to time differences between the server and the client, sometimes this method may not work correctly. To be sure that the deck is not cached, that is, that it expires immediately, set the property to a value of `–1`.

Be aware that cache control is at the deck level, rather than the card level. Consider the following example:

```
<% 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>
          Time in card 1 is now <% =time %>
          <do type="accept" label="Next">
             <go href="#card2"/>
          </do>
      </p>
   </card>

   <card id="card2" title="NoCache">
      <p>
          Time in card 2 is now <% =time %>
      </p>
   </card>
</wml>
```

Although we used the `Response.Expires` property to disable caching, the caching control applies to the deck, not individual cards. This can be seen by the fact that both cards will display the same time (figure 19.22). The time that is inserted is the time the ASP parser interprets the script.


Time in card 1 is now
8:44:00 PM

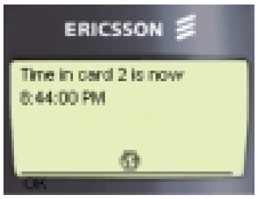

Time in card 2 is now
8:44:00 PM

Figure 19.22
Two cards, same time

META ELEMENT Caching can also be implemented using the `<meta>` element. For examples on caching using the `<meta>` element, refer to chapter 3.

19.12.2 Caching on WAP gateways, proxy servers

To control caching on a proxy server, use the `Response.CacheControl` property.

To prevent proxy servers from caching:

```
<% Response.ContentType = "text/vnd.wap.wml"
   Response.CacheControl = "Private" %>
<?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>
       This deck is not cached by proxy servers.
   </p>
</card>
</wml>
```

Indicates that the ASP document should not be cached

To enable proxy server caching, set the `Response.CacheControl` property to `Public`.

```
Response.CacheControl = "Public"
```

Controlling caching on WAP gateways is the same as for proxy servers. 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."

19.13 SUMMARY

This chapter has touched on creating dynamic WAP applications using ASP. While developing WAP applications is similar to developing web applications, there are a number of points to watch. Detecting the WAP device type and caching are two important topics for any WAP developers.

In part VII of this book, we will look at using Java servlets to develop dynamic WAP applications.