Borland®

Creating WAP Applications with Borland® Delphi™

For Dynamic Data Driven Applications Beyond the Web Browser

by Jani Järvinen

Table of Contents

WAP Overview	2
The history of WAP	2
Who is behind WAP	2
Why develop WAP applications?	2
The technology behind WAP	3
The basics	3
Functions of a gateway	4
WML decks and cards	4
What is a microbrowser?	5
Creating your own WAP applications	5
What hardware and software do I need?	6
WeBroker™ 101	6
Getting started with CGI applications	6
ISAPI Applications and Database sessions	7
Creating actions	8
Responding to actions	9
The initial WML page	10
The ShowSymbol action	10
Creating bitmap images on the fly	11
Testing the example application	12
Conclusion	15

Introduction

The number of handheld Internet-ready devices is growing at an enormous rate. The current estimates are that by the end of year 2002, there will be as many people accessing the Internet wirelessly as there are accessing it traditionally using a desktop computer.

This growth shows no signs of slowing down. New models of mobile phones, personal digital assistants and wireless communications equipment are announced almost daily. Since forthcoming wireless devices are cheaper and easier to use than the regular desktop PC, the preferred communication method in the future will be wireless.

Although there are many competing methods for wireless Internet connection on the marketplace, only Wireless Application Protocol (WAP) has gained global acceptance. WAP is an open, free standard that is supported by all important mobile phone vendors.

"[WAP is] the de facto worldwide standard for providing Internet communications and advanced telephony services on digital mobile phones, pagers, personal digital assistants and other wireless terminals."

- The WAP Forum

Delphi™

white paper

WAP Overview

The history of WAP

The WAP standard has been developed by the WAP Forum, an industry association having more than 500 members. According to the WAP Forum, their members represent of 95% of the global market share and over 200 million subscribers. Companies such as Nokia, Motorola and Ericsson are all members of the forum.

Originally, the WAP Forum consisted of only four members. Since 1997, when the forum was founded, many standards specifications have been released. The newest WAP specification is already in version 1.2, but the most current generation of mobile phones supports version 1.1.

When WAP and mobile phones first came into public attention, the possibilities of the technology immediately raised global interest. Since then, many mobile phone models have been released, and the WAP standard has proceeded to better support the demands of the users.

Who is behind WAP?

WAP is the product of the WAP Forum (www.wapforum.org), an association founded in 1997 by Ericson, Motorola, Nokia, and Phone.com (formerly Unwired Planet). The WAP Forum now has over 200 members and represents over 95 percent of the global handset market.
The primary goal of the WAP Forum is "to bring together companies from all segments of the wireless industry value chain to ensure product interoperability and growth of wireless market."

Why develop WAP applications?

Even though WAP is a new standard, there are already millions of potential customers, both corporate and consumer, waiting for interesting applications to emerge. WAP, being open and secure, is well suited for many different applications, including but not limited to stock market information, weather forecasts, enterprise data, and games.

Certain applications will be accessible only by using WAP phones, but other application types will also benefit from WAP. For example, monitoring and reporting applications could publish data to a WAP phone, giving the user access to the same information regardless of his or her location.
Despite the common misconception, developing WAP applications requires only a few modifications to existing web applications. The current set of web application development tools will easily support WAP development, and in the future more development tools will be announced.

Borland is committed to supporting WAP application development. All its premier development tools, Delphi™, C++Builder™ and Jbuilder™ can be used to develop real-world WAP applications. Using solid Borland tools like these are a great help to developers.

The technology behind WAP

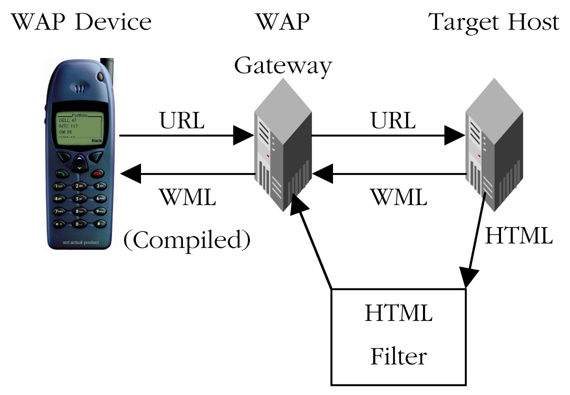
The WAP protocol can be thought as of a collection of different protocols that work together to achieve a common goal. Some of these protocols are already well-known Internet protocols, such as TCP, IP and HTTP. On the wireless side, WAP uses a different set of protocols to transfer presentation data "on the air".

Layer	WAP	Web
Application Layer	Wireless Application Environment (WML and WMLScript)	HTML, Scripting languages
Transport Layer	WSP, WTP, WTLS and WDP	HTTP, SSL, TCP, UDP
Network Layer	Bearer	IP

The basics

In conventional web architecture, a web browser establishes an HTTP connection with a web server, which processes the request and then returns HTML code to the web browser. The web browser will then display the HTML code on screen, along with any images, sounds and animation.

In WAP, the idea is essentially the same, except that technical reasons make the architecture somewhat more complex. The following figure illustrates this.



When a WAP phone user wants to access the Internet, the phone will first initiate a connection with an intermediate server, known as the WAP gateway. The WAP architecture doesn't mandate the use of any specific wireless network type, but instead relies on existing transports, such as TDMA, GSM or CDMA.

Since the gateway is wired directly to the Internet, it will access an existing web server using normal protocols, TCP/IP and HTTP. Of course, the gateway could also be a wireless device, and when learning WAP, is it convenient to think that the link between the WAP phone and the gateway is wireless, while all other connections use cables.

Since all resources are addressed using URLs, the gateway will access the resource on the Internet on behalf of the mobile phone. Thus, all web servers connected to the Internet can be potential WAP application servers.

Because a WAP phone is limited in memory, display technology and transfer speed, a WAP phone doesn't directly display HTML content and images. Instead, a WAP phone is only capable of displaying data coded in the Wireless Markup Language (WML).

This language is an XML-based language that looks like HTML used in normal web applications. Because of the limitations of WML compared to HTML, the web server should return WML code to the gateway. In certain cases, the gateway can convert regular HTML to WML, but this solution should generally be avoided.

Once the gateway has read the WML code from the Web server it will compress it to a binary format, and then transfer it to the mobile phone. The user can then read the information and act accordingly.

Functions of a gateway

A common question among beginning WAP developers is whether a WAP gateway is really required. Technically, the functionality of the WAP gateway cannot be completely eliminated, but it could be integrated into the web server.

Normally, a service provider (operator), such as Bell Atlantic, France Telecom, Sonera or AT&T, hosts the gateway. Since the WAP phone will first initiate a connection the gateway, the gateway can provide additional services to the mobile phone user. For example, the initial welcome page could display a personalized menu and integrate into the other services provided by the operator.

Speaking of the gateway, the operator can also efficiently limit those services that the user can access with his or her mobile phone. As an example, some service providers will limit the user to only those services provided by the service provider.

This is without doubt one of the biggest complaints about the WAP technology. Technically, there is nothing to limit the services a WAP phone can access, and there are a lot of free WAP services on the Internet.

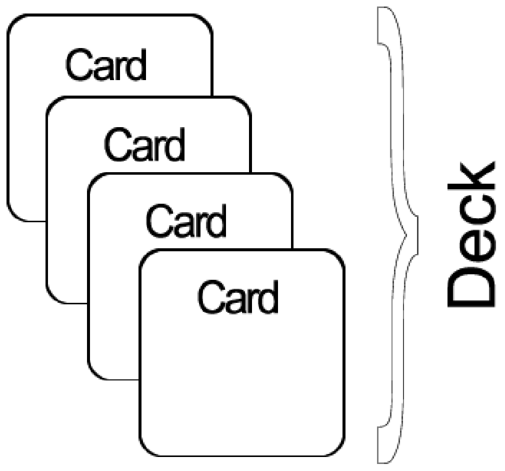
When some mobile phone operators disable those services on the gateway, they limit the number of available services dramatically. This has caused some WAP application developers to provide their own gateways with unlimited access to the Internet.

However, the ability to limit user's possibilities can be considered an advantage. Corporate intranets will without doubt benefit from the ability to limit their users to only certain services.

WML decks and cards

As noted in the introduction, WAP protocol uses WML code to represent the user interface on a WAP phone. This code looks very similar to HTML, but as an XML based language, the format is much more strict. For example, forgetting to close a <b> (bold) tag will cause an error message to be displayed on the mobile phone. In HTML, such errors are silently ignored.

Because of the limitations in bandwidth, screen resolution and user interface, a WML file is organized differently than an HTML file. In WML, each file is divided into *cards*, and a collection of cards is called a *deck*.



The WAP phone is only capable of displaying only a single card at a time. Although a card can be larger than what will fit on the phone's screen, they are generally designed so that they fit completely on the phone's screen.

Navigation between the cards is provided using regular URLs, and transitions between the cards are swift because no connection to the gateway is required. Just like a normal web browser, a WAP phone can also provide a "Back" button with which the user can return to a previous card.

Given the architecture of a WAP application, it is quite evident that navigating between WML decks is an expensive operation, in terms of performance. To compensate for this, WAP phones support resource caching as well as a primitive scripting language, WMLScript.
This scripting language can be used to manipulate the properties of the microbrowser. One good use of WMLScript is form validation. Used this way, the WMLScript can help to keep the performance of the application high.

What is a microbrowser?

Just like an ordinary PC requires a web browser to display web pages, so does a WAP require a browser to display the WML cards. Because of the memory requirements, the browser in a WAP phone is called a *microbrowser*. Although tiny in memory footprint, it supports many features and is even scriptable.

The web server

In the WAP architecture, the web server communicates with the WAP gateway, accepting HTTP requests and returning WML code to the gateway. The HTTP protocol mandates that each reply must include something called a MIME type (Multi-Purpose Internet Mail Extensions).

In normal web applications, this MIME type is set to text/html, designating normal HTML code. Images, on the other hand, could be specified as image/gif or image/jpeg, for instance. With this *content type* specification, the web browser knows the data type that the web server returns.

In WAP applications those previously mentioned, MIME types cannot be used. Instead, a new set of MIME types must be used, as shown in the following table:

File type	MIME type
WML (.wml)	text/vnd.wap.wml
WMLScript (.wmls)	text/vmd.wap.wmlscript
WBMP (.wbmp)	image/vnd.wap.wbmp

In dynamic applications, the MIME type must be set on the fly, whereas in static WAP applications the web server must be configured appropriately. Forgetting to properly set the MIME type is probably the most common mistake in beginning WAP application development.
For more information about configuring MIME types for your web server, please consult your web server documentation.

Creating your own WAP applications

Borland® Delphi™ already includes excellent support for building web applications. With only a little additional work, it is possible to create advanced WAP applications. However, while normal web applications can be developed only with Delphi, a web browser and a web server, developing WAP applications requires additional software and hardware.

When developing WAP applications, it is generally in the interest of the application developer to re-use code and components from previous applications. With careful design and usage of technologies such as XML, it is possible to create an application that will serve both ordinary web browsers (HTML) as well as WAP microbrowsers (WML).

What hardware and software do I need?

At minimum, developing WAP applications requires a web server and a WAP simulator. Using simulator software while developing a WAP application is convenient as all the required software can be installed on the development PC— the one on which Delphi is installed.

Although software simulators are good in their own right, no WAP application should go into production without testing it with actual hardware. The following list gives a quick overview of the necessary hardware and software to test and develop WAP applications:

- a Web server with connection to the Internet
- a WAP simulator
- a WAP gateway
- a WAP phone

The Appendix A lists some WAP gateway, simulator and phone vendors. For the purposes of this article, it is assumed that Microsoft® Internet Information Server 5.0 is used as the web server and Nokia™ WAP Toolkit version 2.0 as the WAP simulator.

WebBroker™ 101

In Delphi, web application development is done using a technology called WebBroker.™ This technology gives the software developer freedom of choice over web application architecture while still maintaining a high level of abstraction over the underlying technologies. The WebBroker technology is available in Delphi 5 Professional and Enterprise versions. It is not available in the Standard version of Delphi 5. In the future, the WebBroker technology will also be available on the Linux platform in the forthcoming Borland Kylix™, the native Rapid Application Development (RAD) environment for Linux.®

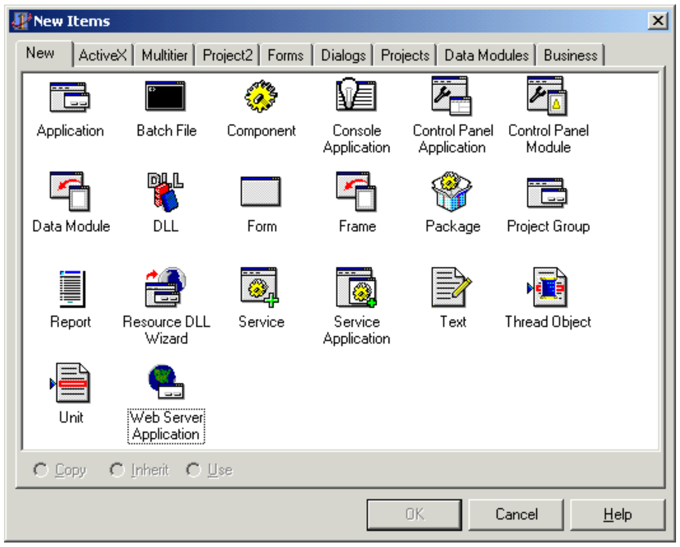
At present, WebBroker supports CGI, WinCGI, ISAPI and NSAPI applications. The application type is selected when creating a WebBroker application and stays the same until a new application is created. CGI and WinCGI applications result in an EXE file, and the ISAPI and NSAPI applications result in a DLL file being created.

All application types are created equally, and the source code is almost always compatible with other application types. Thus, if the application type needs to be changed at a later time, it's simply a matter of copying and pasting the code to a new application.
In the Delphi IDE, WebBroker applications open up a *web module*, which is very similar to a data module. The web module is the container for database components and other non-visual components that are required by the web application.

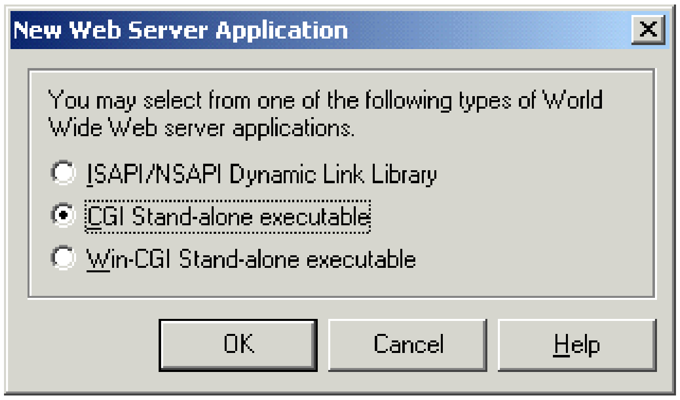
As noted previously, building a WAP application only requires a few modifications to existing web applications. The following sections demonstrate how the Delphi WebBroker technology can be used to build real WAP applications.

Getting started with a CGI application

Developing WebBroker applications begins by selecting the Web Server Application from the New Items dialog box (activated using the File / New menu command).



After clicking OK, Delphi displays the New Web Server Application dialog box. Here, you may choose the application type you wish to use. This document illustrates how to create a conventional CGI application, but you can choose another application type as well.


New Web Server Application

You may select from one of the following types of World Wide Web server applications.

- ( ) ISAPI/NSAPI Dynamic Link Library
- (●) CGI Stand-alone executable
- ( ) Win-CGI Stand-alone executable

For example, choosing an ISAPI application improves the performance of your application if you are using Microsoft IIS as your web server. Choosing a CGI application will create an application that is most compatible with the forthcoming version of Delphi on the Linux platform, as ISAPI DLLs are not supported on the Linux platform.
Once the correct application type has been chosen, Delphi will display a blank web module on the screen.

The example application presented in this article is a classic stock market application, one that has the ability to display stock pricing and draw history graphs.

The example application uses the DBDEMOS sample database that ships with Delphi 5. If you haven't already done so, install the sample databases from Delphi's installation CD, and make sure the Borland Database Engine (BDE) alias DBDEMOS points to that data. The easiest way to do this is to use the Custom Setup option in Delphi's installation program. Alternatively, you can copy the necessary files from the RunImage directory on the installation CD and create the alias manually in BDE Administrator.

ISAPI applications and database sessions

If you are building ISAPI or NSAPI web applications, you need to address threading issues in your applications. Every ISAPI/NSAPI web application is a DLL, which is loaded into memory by the web server. By default, the DLL stays in memory until the web server is stopped.

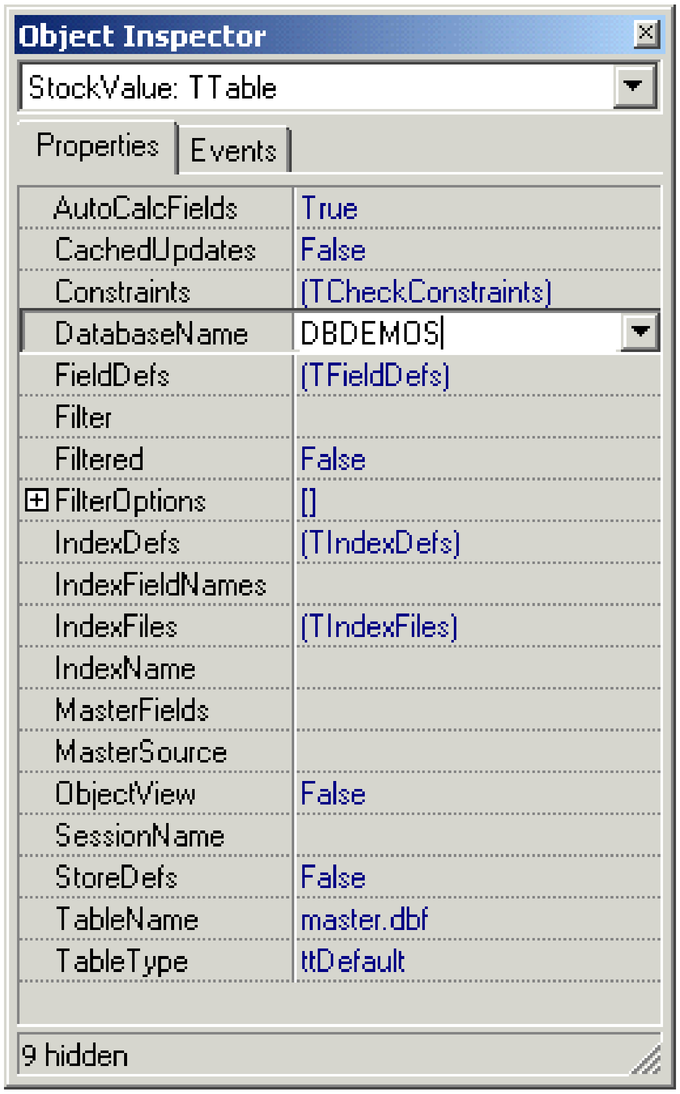
When multiple, simultaneous hits occur to your DLL, the web server spawns threads to call your DLL. This means that your application's code is potentially running in the context of different threads. To protect against possible conflicts, you must protect your code using synchronization objects, as appropriate.

Other than protecting global variable access etc., you must also protect your database connections if you are using BDE enabled datasets. However, this is easier than it seems at first.

To protect BDE datasets against multi-threading issues, you must use a TSession component. A TSession component is connected to a TTable or TQuery component using the SessionName property. You should also set the AutoSessionName property to True on the TSession component. This automatically creates a unique session name at run-time.

Note that it is not necessary to use a TSession component in a CGI application, since only one thread is running at a time. For more information about the TSession component and its use in web applications, consult Delphi's on-line Help.

Because a database is needed for the example application, you should begin by dropping a TTable component onto the web module. Connect it to the master.dbf file on the DBDEMOS alias by using the Object Inspector™ to select DBDEMOS from the DatabaseName property, and then select master.dbf from the TableName drop down. After this has been done, you are ready to proceed with creating actions for your WAP application.



AutoCalcFields	True
CachedUpdates	False
Constraints	(TCheckConstraints)
DatabaseName	DBDEMOS
FieldDefs	(TFieldDefs)
Filter	
Filtered	False
⊞ FilterOptions	[]
IndexDefs	(TIndexDefs)
IndexFieldNames	
IndexFiles	(TIndexFiles)
IndexName	
MasterFields	
MasterSource	
ObjectView	False
SessionName	
StoreDefs	False
TableName	master.dbf
TableType	ttDefault

Creating actions

Just like an ordinary desktop application can have a menu to let the user choose the different commands supported by the application, so can a WebBroker application respond to different commands.
In WebBroker terminology, these different commands are called *actions*, and each WebBroker application can have an unlimited number of actions. To understand how actions work, consider the following URLs that point to a hypothetical web application webapp.exe:

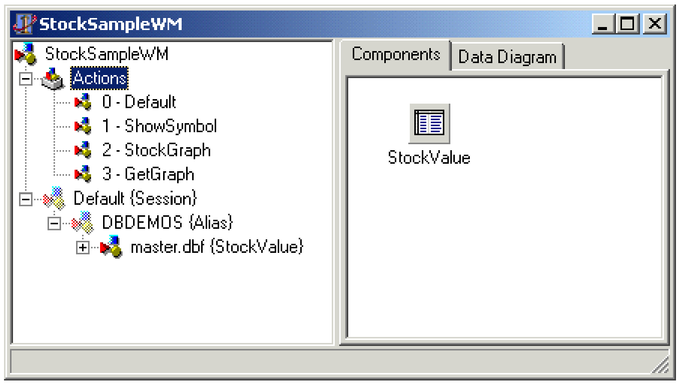
http://www.somehost.com/scripts/webapp.exe/oranges

http://www.somehost.com/scripts/webapp.exe/bananas

Here, the URL is appended with a *path*, telling the web application the fruit the user is interested in. In WebBroker applications, each path is normally handled by one action. This is also the case with the example application.

No matter what happens to your application, it should always return meaningful data to the user, even if it can only be an error message. This is best accomplished by creating a *default action*, which will be run if no other action handles the request.

To create such an action, right-click the Action tree node in the left-hand side of the web module, and choose Add Item from the menu. It is convenient to rename the actions so that it is easy to find a particular action in a more complex application.



Responding to actions

What an action does is determined by the code that you place in its OnAction event handler. In the case of the sample application, the OnAction event handler of the default action looks like this:

```
procedure
TStockSampleWM.StockSampleWMDefaultAction(Sender:
TObject;
  Request: TWebRequest; Response: TWebResponse;
  var Handled: Boolean);
begin
  Response.ContentType := WML_ContentType;
  Response.Content :=
StringReplace(WML_InvalidCall,'%datetime%',
                                    DateTimeToStr(Now),[]);
end;
```

For WAP applications, the most important point is to set the correct MIME type before returning the WML code. In the above example, the action sets the MIME type by setting the Response.ContentType property to WML_ContentType. This constant is defined in the example application as:

```
Const
  WML_ContentType = 'text/vnd.wap.wml';
```

The second line of code makes the action return the following WML code:

```
<?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="problem" title="StockSample">
    <p>Invalid call at %datetime%.
      <do type="prev" label="Back">
        <prev/>
      </do>
    </p>
  </card>
</wml>
```

Of course, the string “%datetime%” is replaced by the current data and time at runtime.

The initial WML page

Although the sample application presented in this white paper is mostly dynamic, it does contain two static files that are required for its operation. The most important is the main screen file, contained in stocksample.wml; the other is simply a wireless bitmap (WBMP) file.

The content of the static WML file is shown in the following listing:

```
<?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="init" newcontext="true">
    <onevent type="ontimer">
      <go href="#mainscreen"/>
    </onevent>
    <timer value="30"/>
    <p align="center">
      <b>StockSample</b>
      <br/>
      <img src="logo.wbmp" alt="logo"/>
      <br/>
      <small>Version 1.0</small>
    </p>
  </card>
  <card id="mainscreen" title="Main Screen"
newcontext="true">
    <do type="accept">
      <go href="/scripts/stocksample.exe/showsymbol"
method="get">
        <postfield name="symbol" value="$(symbol)"/>
      </go>
    </do>
    <p>
      <b>Enter stock symbol:</b>
      <br/>
      <input name="symbol" format="*A" maxlength="4"
       emptyok="false"/>
    </p>
  </card>
</wml>
```

The file contains two WML cards, identified as “init” and “mainscreen.” The first card displays a logo screen and includes a timer. This timer fires after three seconds, and the main screen is then automatically displayed.

The main screen allows the user to enter a stock symbol and follow a link to display information about the stock symbol in question. Once the user has entered the symbol and chosen to accept the form, control is transferred to the example application, stocksample.exe. The path “/showsymbol” corresponds to the ShowSymbol action on the web module.

The ShowSymbol action

The code for the ShowSymbol action is as follows:

```
procedure
TStockSampleWM.StockSampleWMShowSymbolAction(
  Sender: TObject; Request: TWebRequest; Response:
TWebResponse;
  var Handled: Boolean);
Var Symbol,WML : String;
begin
  Response.ContentType := WML_ContentType;
  Symbol := Request.QueryFields.Values['symbol'];
  Try
    StockValue.Open;
    If StockValue.Locate('SYMBOL',Symbol,[]) Then Begin
      WML := StringReplace(WML_SymbolInfo,'%symbol%',
                    Symbol,[rfReplaceAll]);
      WML := StringReplace(WML,'%price%',
                    StockValueCur_Price.AsString,[]);
      WML := StringReplace(WML,'%high%',
                    StockValueYrl_High.AsString,[]);
      WML := StringReplace(WML,'%low%',
                    StockValueYrl_Low.AsString,[]);
      WML := StringReplace(WML,'%rating%',
                    StockValueRating.AsString,[]);
      WML := StringReplace(WML,'%rec%',
                    StockValueRcmndation.AsString,[]);
      Response.Content := WML;
    End
    Else Begin
      If (Symbol = '') Then Symbol := '(empty)';
      Response.Content :=
StringReplace(WML_SymbolNotFound,
        '%symbol%',Symbol,[]);
    End;
  Finally
    StockValue.Close;
  End;
end;
```

Here, the stock symbol entered by the user is read from the QueryFields property of the Request object, corresponding to a HTTP GET variable, as specified on the stocksample.wml file. The code then tries to locate the stock symbol in the example database, and if found, formats the parameterized WML code accordingly.

If the symbol is not found, an error message is displayed. Note how the MIME type is set at the beginning and how the database connection is protected using a Try/Finally clause. It is important to minimize all error conditions that cause memory leaks in WebBroker applications.

Creating bitmap images on the fly

WAP phones provide support for primitive monochrome bitmaps, known as wireless bitmaps (WBMPs). These files have a simple internal representation (level 0), which makes it possible to create these images dynamically on the fly.

When the user has selected a stock symbol and chosen to display the current pricing, the sample application provides a link to display a history graph about the symbol's performance. The sample application is able to draw this graph in four variations, depending on the required time period (3, 6, 9 or 12 months).

This selection is supported by the following WML code snippet:

```
<br/><anchor>Stock Graph
  <go href="/scripts/stocksample.exe/stockgraph"
method="get">
    <postfield name="symbol" value="%symbol%"/>
    <postfield name="period" value="$(period)"/>
  </go>
</anchor>:<br/>
<select name="period" value="3" title="Graph:">
  <option value="3">3 months</option>
  <option value="6">6 months</option>
  <option value="9">9 months</option>
  <option value="12">12 months</option>
</select>
```

When the user chooses to display a graph, the control is transferred to the OnAction event handler of the StockGraph action on the web module. The action outputs WML code that includes the stock symbol name along with the following WML code tag:

```
<img
src="/scripts/stocksample.exe/getgraph?s=%symbol%&am
p;p=%period%" alt="graph"/>
```

The tag tells the WAP phone that it should display an image, which can be retrieved from aURL specified by the SRC parameter on the tag. Since the URL points to the sample application, the following event handler code gets executed:

```
procedure
TStockSampleWM.StockSampleWMGetGraphAction(Sende
r: TObject;
  Request: TWebRequest; Response: TWebResponse;
  var Handled: Boolean);
Var MemStrm : TMemoryStream;
begin
  MemStrm := TMemoryStream.Create;
  CreateWirelessBitmap(MemStrm);
  MemStrm.Position := 0;
  Response.ContentType := WBMP_ContentType;
  Response.ContentStream := MemStrm;
  Response.SendResponse;
end;
```

Since a bitmap image is pure binary data, a simple string variable cannot be used to hold the data. Instead, the example action uses a memory stream for this purpose, as streams are readily supported by the WebBroker architecture.

It is again important to note that the MIME type must be set correctly before returning the bitmap. This time the MIME type is set to “image/vnd.wap.wbmp”. Also note that it is not necessary to free the memory stream explicitly – this is done by the SendResponse method of the Response object.

Without delving too much into the file format of WBMPs, the CreateWirelessBitmap method looks like this:

```
Procedure
TStockSampleWM.CreateWirelessBitmap(MemStrm :
TMemoryStream);
Const Header : Array[0..3] of Char = #0#0#104#20;
Var
  Bmp : Array[1..104,1..20] of Boolean;
  X,Y : Integer;
  Dir : Integer;
  Bit : Integer;
  B   : Byte;

Begin
  { clear the bitmap out }
  FillChar(Bmp,SizeOf(Bmp),0);
  { draw X and Y axis }
  For X := 1 to 104 do Bmp[X,20] := True;
  For Y := 1 to 20 do Bmp[1,Y] := True;
  { draw random data }
  Randomize;
  Y := Random(20)+1;
  Dir := Random(10);
  For X := 1 to 104 do Begin
    Bmp[X,Y] := True;
    If (Dir > 4) Then Y := Y+Random(2)+1
    Else Y := Y-Random(2)-1;
    If (Y > 20) Then Y := 20;
    If (Y < 1) Then Y := 1;
    Dir := Random(10);
  End;
  { create WBMP data }
  MemStrm.Write(Header,SizeOf(Header));
  Bit := 7; B := 0;
  For Y := 1 to 20 do Begin
    For X := 1 to 104 do Begin
      If (Bmp[X,Y] = True) Then B := B Or (1 shl Bit);
      Dec(Bit);
      If (Bit < 0) Then Begin
        B := Not B;
        MemStrm.Write(B,SizeOf(B));
        Bit := 7;
        B := 0;
      End;
    End;
  End;
End;
```

Please note that the method will create a 104 x 20 bitmap with random data. Random data must be used because the DBDEMOS example tables don’t contain stock history data.

Testing the example application

Since setting up a WAP gateway and configuring a real WAP phone would warrant yet another white paper, this document uses the Nokia WAP Toolkit to demonstrate the workings of the example application. The Nokia toolkit is an efficient phone simulator, based on Java technology that can be downloaded freely from the Nokia developer site at www.forum.nokia.com after registration (there are also other simulators available, please see Appendix A for further details).

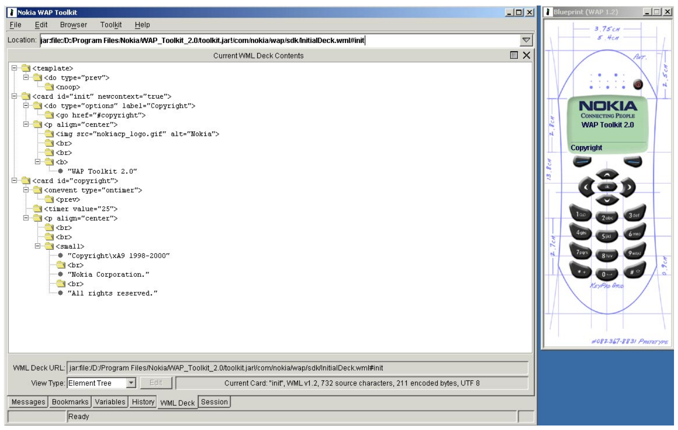
The current version of the Nokia WAP Toolkit, 2.0, supports two phone models, Nokia 7110 and a “blueprint” model. The latter is set up as the default, so using it is convenient. Before the simulator can be used though, the example application’s files need to be copied to the web server.

In the case of Microsoft IIS 5.0, the files should be copied as follows:

File	Location
stocksample.wml	\inetpub\wwwroot
logo.wbmp	\inetpub\wwwroot
stocksample.exe	\inetpub\scripts

Note: Borland Database Engine (BDE) and the sample database must also be installed on the web server if it is a different computer from the one you are doing your development work with. This may require setting Windows NT® or Windows 2000 access permissions for the IUSR_machine user account.

Once all the necessary files are in their correct locations, the WAP simulator can be started. The default setup looks like the following:


Nokia WAP Toolkit
File Edit Browser Toolkit Help
Current WML Deck Contents
Messages Bookmarks Variables History WML Deck Session
Ready
Blueprint (WAP 1.2)
NOKIA
Connecting People
WAP Toolkit 2.0
Copyright

Location: jar:file:D:/Program Files/Nokia/WAP_Toolkit_2.0/toolkit.jar!/com/nokia/wap/sdk/InitialDeck.wml#init

```
<template>
  <do type="prev">
    <noop>
<card id="init" newcontext="true">
  <do type="options" label="Copyright">
    <go href="#copyright">
  <p align="center">
    <img src="nokiacp_logo.gif" alt="Nokia">
    <br>
    <br>
    <b>
      "WAP Toolkit 2.0"
<card id="copyright">
  <onevent type="ontimer">
    <prev>
  <timer value="25">
  <p align="center">
    <br>
    <br>
    <small>
      "Copyright\xA9 1998-2000"
      <br>
      "Nokia Corporation."
      <br>
      "All rights reserved."
```

WML Deck URL: jar:file:/D:/Program Files/Nokia/WAP_Toolkit_2.0/toolkit.jar!/com/nokia/wap/sdk/InitialDeck.wml#init

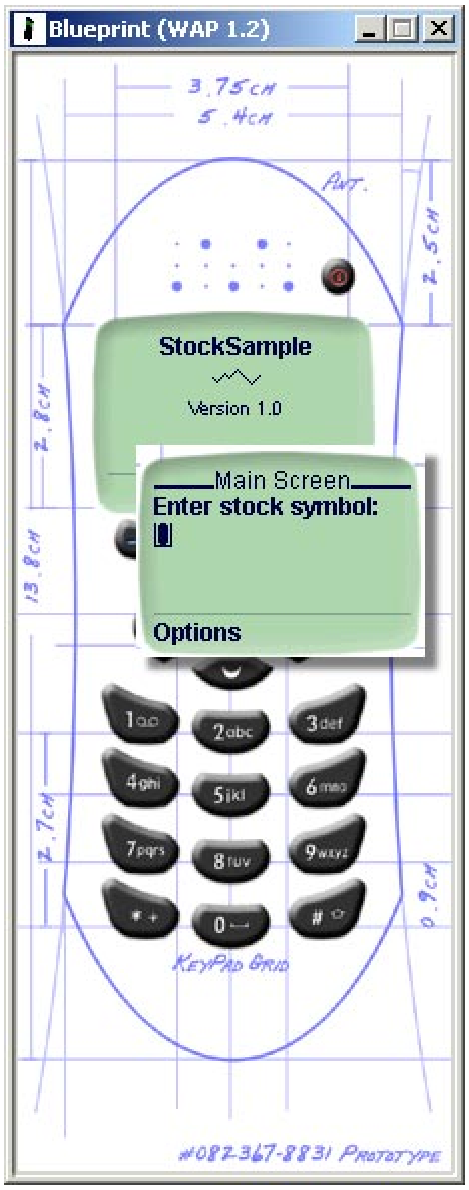
View Type: Element Tree

Current Card: "init", WML v1.2, 732 source characters, 211 encoded bytes, UTF 8

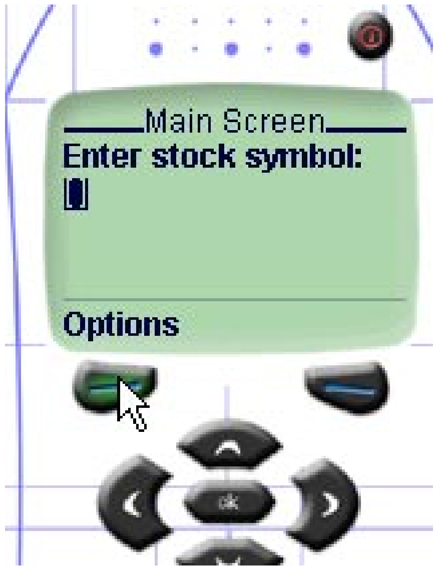
To start running the example application, choose the Load Location command from the simulator’s Browser menu. As the URL, enter:

http://localhost/stocksample.wml

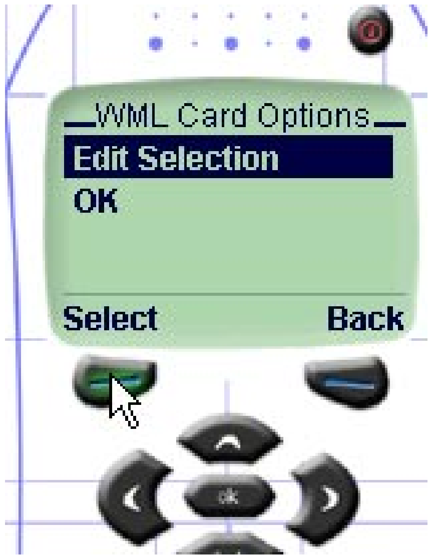
After clicking the OK button, the simulator will start to load the WML card deck, first briefly displaying the welcome screen and then the main menu, as shown here:



To specify the stock symbol you wish to view, click the small button on the virtual phone just under the Option text.



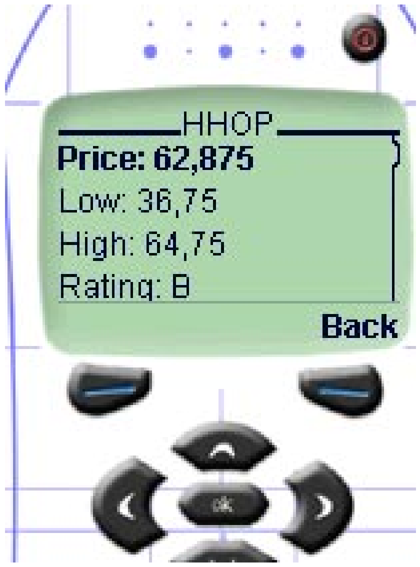
From the menu, choose Edit Selection. If required, use the tiny arrow button on the phone to make your selection.



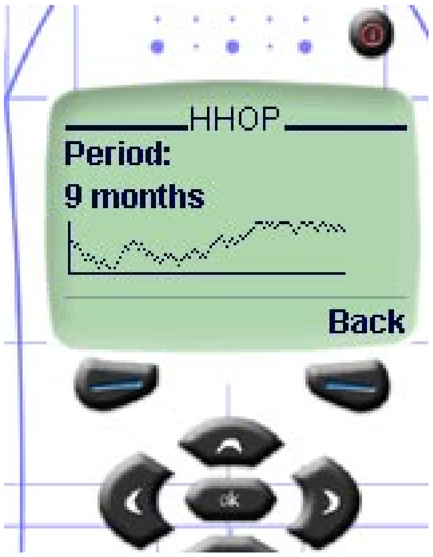
Then, enter one of the following supported stock symbols:

- UIN
- SCP
- HHOP
- USMD

After entering the stock symbol, choose OK twice to accept the symbol and then to proceed to the symbol page. Remember, the symbol page will be dynamically generated by the Delphi application. The stock data will be fetched from the dBase® database.



Continuing with another example, a 9-month graph gives the following output (you might need to scroll the screen to see the image selection menu):



To refresh a new graph (since it was random data), use the simulator's Browser / Refresh Card command. Using Browser / Reload Deck will also work, but this will cause you to lose the Back command on the microbrowser. To return to the original page, use the Load Location command.

Conclusion

WAP is an exciting new technology providing many interesting possibilities to application developers. Since WAP has already gained the necessary popularity, developers can begin to provide WAP solutions to customers immediately.

The current set of tools also provides good support for WAP application development. Borland Delphi can be used to build WAP applications with its flexible WebBroker architecture, and, beginning development doesn't require expensive hardware, since initial testing can be done with free or low-cost software.

Because of the possibilities, developers are recommended to take advantage of the current mobile terminal possibilities. In the future, the possibilities will be even more amazing.

Glossary

Acronym	**Description**
API	Application Programming Interface
BDE	Borland® Database Engine
CDMA	Code Division Multiple Access
CGI	Common Gateway Interface
DTD	Document Type Definition
GSM	Global System for Mobile communications
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IIS	Internet Information Server
IP	Internet Protocol
ISAPI	Internet Server API
MIME	Multi-purpose Internet Mail Extensions
NSAPI	Netscape Server API
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WBMP	Wireless Bitmap
WML	Wireless Markup Language
WSP	Wireless Session Protocol
WTLS	Wireless Transport Layer Security
WTP	Wireless Transport Protocol
XML	Extended Markup Language

Appendix A

A list of interesting and valuable links for WAP application developers.

Borland® Delphi™
http://www.borland.com/delphi/

WAP Forum
http://www.wapforum.org

WAP technical specifications
http://www.wapforum.org/what/technical.htm

WAP simulators
http://forum.nokia.com

http://developer.phone.com

http://www.winwap.org

WAP gateways
http://www.waplite.com

http://www.realwow.com

WAP phone vendors
http://www.nokia.com

http://www.ericsson.com

http://www.motorola.com

http://www.siemens.com

Other interesting links
http://www.anywhereyougo.com

http://www.ericsson.com/developerszone/

http://www.delphizine.com/features/2000/04/di200004jj_f
/di200004jj_f.asp

Jani Järvinen works as a technical support person for Borland products in Finland. He is also a local Delphi trainer and freelance author. He has written to magazines like Delphi Informant, The Delphi Magazine and Delphi Developer. He specializes in Windows API and Internet technologies. He is a Microsoft Certified Professional (MCP) and a co-author of a Finnish Delphi book aimed at the professional developer.

Borland®

100 Enterprise Way
Scotts Valley, CA 95066-3249
www.borland.com | 831-431-1000