CHAPTER

24

Wireless Development

IN THIS CHAPTER

Without a doubt, two technologies in the past 10 years have touched our lives more than any others: the Internet and mobile devices. Despite the ups and downs of Internet-based businesses, the Internet—and the Web in particular—has permanently changed our lives. It has affected the way we communicate, shop, work, and play. What's more, many of us now shudder at the thought of being caught somewhere without our trusty mobile phone or Personal Digital Assistant (PDA). Given their importance in our lives, it seems natural that these two technologies are now in a state of convergence, with mobile devices becoming wireless tentacles reaching out from the wired Internet to provide us with the services and information to which we've become so addicted.

With mobile information devices becoming more necessity than novelty in today's business and social climate, we developers are faced with the challenge of leveraging this hardware and infrastructure in order to fulfill the ever-growing demand to push data and applications out to mobile devices. With a dizzying array of mobile devices, networks, and technologies on the market, the key questions for developers become: Which of the wireless platforms should you target? What is the most efficient way to target them? What technologies can I leverage to mobilize my data and applications? What are the trade-offs between all these platforms and technologies?

This chapter is by no means intended to serve as an exhaustive how-to, describing how to implement all the various mobile technologies. That would require volumes. We will, however, have done our job if you get two things from this chapter. First, you will hopefully be able to use this chapter as a *cheat sheet* in understanding the role many of the various types of hardware, software, and technologies play in mobile computing from a developer's perspective. Second, you should understand how some of these mobile technologies can be implemented using Delphi.

Evolution of Development—How Did We Get Here?

Before discussing how you might build the applications to harness these emerging trends in information technology, it's important to look back at what brought us here. Here is a rather simplified snapshot of recent trends in information technology.

Pre-1980s: Here There Be Dragons

Before the PC revolution of the 1980s brought information technology to the masses, development for these systems was a jumble of mainframes, terminals, and proprietary systems. Developer tools were generally rudimentary, making application development an expensive and time-consuming process reserved for true bit-heads.

Late 1980s: Desktop Database Applications

After the PC revolution took hold, folks began to leverage the new found power residing on their desktops using desktop database applications such as dBASE, FoxPro, and Paradox. General application development tools also became more mature, making application development a relatively straightforward task using third generation languages such as C, Pascal, and BASIC. DOS was king of the desktop, providing applications with a common platform upon which to build. Local area networks were becoming practical for businesses of all sizes, which provided for centralized storage of data on file servers.

Early 1990s: Client/Server

Corporate networks were now taken for granted; most everyone in the office was connected. The question now was how to bridge the gap between the aging mainframe systems and the non-scalable desktop databases that were both important to business. The answer was client/server systems, the notion of powerful databases from companies such as Oracle, Sybase, and Informix connected to user-interfaces running on PCs. This enabled systems to leverage the power on every desktop while enabling database servers to perform their specialized tasks. Fourth generation development tools such as Visual Basic and Delphi made development easier than ever before, and database support was built in as a first class citizen of the tools.

Late 1990s: Multitier and Internet-Based Transactions

The primary problem with the client/server model is the notion of where the business logic should reside—place it on the database server and you limit scalability; place it on the client and you have a maintenance nightmare. Multitier systems solved this problem by placing the business logic on one or more additional *tiers* logically and/or physically separate from the client and server. This enabled properly written systems to scale to a nearly unlimited extent and paved the way for complex transactions to be served to thousands or millions of clients via the Internet. Development tools extended into the multitier world with technologies such as CORBA, EJB, and COM. Businesses were quick to leverage the Internet to offer information and services to employees, clients, and partners, and industries grew up around the ability to manage, publish, and exchange data between machines over the Internet.

Early 2000s: Application Infrastructure Extends to Wireless Mobile Devices

So, what is the net result of the vast information availability provided by the Internet? The answer is two words: information addiction. The availability of information and services via the Internet has made us dependent on the same in ever increasing aspects of our lives. PDAs and mobile phones have served to scratch that itch, feeding our information addiction while away from our desks. Application servers and development tools are growing in scope to manage the push of functionality to these types of devices. The potential market for applications on mobile devices is mind boggling in size because the projected number of these devices coming into the market over the next few years dwarfs the numbers for PCs.

Mobile Wireless Devices

Between mobile phones, PDAs, and smart pagers, there is no shortage of devices from which to choose should you want to remain connected while away from your desk. Those of us who have trouble choosing sometimes carry all three on belts that cause us to resemble Batman more and more everyday. We are also seeing a convergence of these devices into single, multifunctional devices. Recent examples of this include the mobile phone Springboard module for Handspring handhelds, the Kyocera Smartphone running PalmOS, and Microsoft's Stinger Windows CE-powered mobile phone. In this section, we will call out a few of the leaders in this area.

Mobile Phones

Mobile phones are by far the most pervasive variety of mobile wireless device. Mobile phones have moved beyond the realm of pure voice communication systems into the realm of data communications. Most notably, the majority of new phones coming into the market support text messaging using Short Message Service (SMS) and Web-like browsing using Wireless Application Protocol (WAP). Current data rates are rather paltry at 9.6-14.4k, but new technologies promise to deliver speeds of up to 2Mbits within 2-3 years.

PalmOS Devices

Devices running the Palm Computing's PalmOS operating system have been the market share leader in the PDA space for several years. Some PalmOS devices have wireless capability built in (such as the Palm VII series or the Kyocera Smartphone), and wireless can be added to others through the use of a wireless modem (such as those made by Novatel) or a mobile phone connector available from Palm. A wide range of companies have licensed PalmOS from Palm, Inc. for inclusion in their own devices, including Handspring, Sony, Kyocera, Symbol, Nokia, Samsung, and TRG. Advantages of PalmOS includes the fact that they own the overwhelming share of the market for PDAs, and there is strong developer community with an active third-party market.

Pocket PC

Compaq, HP, Casio, and other manufacturers produce PDAs based on Microsoft's Pocket PC (formerly Windows CE) operating system. To date, none of these devices have built-in wireless capability, but they do support wireless modems in a manner similar to PalmOS devices. Even more, Pocket PC devices tend to be a bit more powerful than their PalmOS counterpart, with some having the capability of accepting standard PC Cards (PCMCIA). This potentially allows an even greater range of expansion to higher-bandwidth wireless networks.

RIM BlackBerry

The BlackBerry provides PDA-type functionality in a pager-sized form factor. With an internal wireless modem and type-with-your-thumbs keyboard, the BlackBerry is especially well suited to mobile e-mail tasks. However, the BlackBerry also supports web browsing via a third-party browser. I have found the BlackBerry to be an outstanding platform for corporate e-mail, thanks to built-in integration with MS Exchange or Lotus Domino, but the device is wanting as a Web appliance because of its screen size and navigation capabilities.

Radio Technologies

Radio technologies provide the connection between mobile devices and the Internet or corporate LAN.

GSM, CDMA, and TDMA

These are the primary technologies used as the transport for mobile phones, and they are often referred to as *2G* because they embody the second generation of mobile communications networks (*1G* being analog service). Most networks in the United States are based on CDMA or TDMA, whereas most of the rest of the world relies on GSM. The details of these technologies are relatively unimportant from a software developer's point of view, except to know that the very existence of these competing standards makes it difficult to create applications that function across the spectrum of phones and networks. Generally, data speeds on these types of networks top out at 9.6-14.4k.

CDPD

Cellular Digital Packet Data (CDPD) is a technology that enables packet-based data transfer over wireless networks, offering increase in bandwidth and "always on" functionality. CDPD is common with aftermarket PDA wireless service in the United States, such as that provided by GoAmerica or OmniSky, and speeds reach about 19.2k.

3G

3G, or third generation, mobile networks are designed from the ground up to handle a variety of different types of media streams and boast bandwidth estimated to be somewhere in the range of 384k-2M. The most likely candidates to be the 3G standard bearers are technologies known as EDGE and UMTS. However, although the technology exists and several carriers own enough spectrum to implement 3G networks, no carrier seems to want to be the first to make the multi-billion dollar investment in network upgrades in order to move forward with 3G.

GPRS

General Packet Radio Service (GPRS) is considered the migration path from 2G to 3G, and is often therefore referred to as *2.5G*. GPRS enables packet-based traffic over existing 2G infrastructure with only relatively minor upgrades. Realized throughput on GPRS networks will likely be in the 20-30k range.

Bluetooth

Devices incorporating Bluetooth radio technology are just now beginning to come available in the market. Bluetooth is an important emerging technology because it permits short range, ad-hoc networking among different types of devices. Because Bluetooth radio modules are very small, have low power, and are relatively inexpensive, they will be embedded in all manner of mobile devices, including phones, PDAs, laptops, and so on. Most Bluetooth radios will have a range of about 10 meters and enjoy about 700k of bandwidth. Potential applications for Bluetooth include synchronizing data between PDA and computer when they come into proximity with one another or providing a laptop with Internet connectivity via a mobile phone in one's pocket. A new term, personal area network (PAN), is used to describe this notion of a small wireless network where all of our personal mobile devices regularly communicate with one another.

It's more accurate to think of Bluetooth as a replacement for serial, USB, or IEEE 1394 cables than as a Ethernet-type networking technology. The current iteration of Bluetooth supports only one master device controlling a maximum of seven simultaneous slave devices.

802.11

Although Bluetooth technology is designed as a short-range personal networking technology, 802.11 is intended to be used for LANs. The current generation of this technology, 802.11b or *WiFi*, provides up to 11Mb of bandwidth, with a 45Mb version known as 802.11a on the horizon. 802.11 has a range of about 30 meters, with greater ranges possible using special antennas. 802.11's power requirements are greater than that of Bluetooth, and the devices are larger in size; the radio device can fit inside a standard PC Card, which is great for laptops, but not convenient for phones or most PDAs.

One important note to keep in mind is that Bluetooth and 802.11 share the same 2.4 GHz spectrum, so it is possible that the two might interfere with one another when occupying the same space. Although it's unlikely that they would completely freeze each other out because of the fact that both use spread spectrum technology to hop frequencies many times per second, it's feasible that performance could suffer on either or both connections because of mutual interference.

Server-Based Wireless Data Technologies

Wireless data technologies ride on top of the radio technology in order to provide data and services to mobile devices. These technologies involve servers generating content, which is sent wirelessly to clients and interpreted by built-in software residing on the client.

SMS

Short Message Service (SMS) technology is used to send short (generally 100 to 160 character maximum) text messages to mobile phones. Aside from the limited message length, SMS technology is limited due to issues of interoperability between network operators and varying SMS protocols employed by operators. However, SMS has become very popular—particularly in Europe—because of its ease-of-use and wide availability.

Because each carrier might employ slight variations on the SMS theme, techniques for developing applications that support SMS can vary depending on the carrier you are targeting. Although GSM has an advantage over other mobile phone networks in that the support for SMS is built into the GSM standard, from an application developer's standpoint, it can still be challenging to send SMS messages from a server connected to the Internet to a mobile client. This is because you have to work with SMS servers on the carrier side, which might involve a varying support of standards and even licensing fees.

We recommend one of two avenues for incorporating SMS support into servers. The first option is to simply use e-mail; most carriers support the sending of an SMS message by sending an e-mail message to a specific e-mail address that contains the number of the recipient's phone. Although this is a relatively simple approach from a technical standpoint, the disadvantage is that support isn't universal and it adds another layer of potential failure. The second option is to purchase any one of many third-party tools that handle the sending of SMS messages on a variety of networks. This is the preferred technique, although it will involve some up-front costs and/or licensing fees.

WAP

Wireless Application Protocol (WAP) was established as a standard means for accessing information from the Internet via a mobile device. The general acceptance of WAP in the market has been mixed. On the one hand, WAP has been well received by network operators and phone manufacturers because it was designed from the beginning to work over any wireless service and network standard on practically any device. However, the user experience with WAP hasn't been positive overall because of limitation in display, data entry capabilities, and wireless bandwidth. Additionally, because WAP sites have little way to generate revenue based on usage, there is not a strong business incentive for developing high-quality WAP sites. Content for WAP systems is developed in an XML-based language known as Wireless Markup Language (WML).

The typical WAP application architecture is illustrated in Figure 24.1.

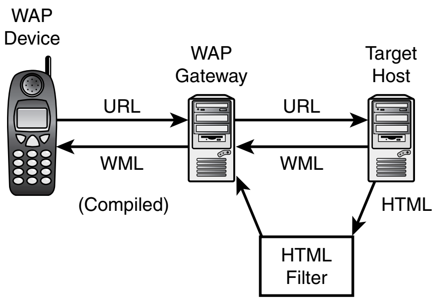


FIGURE 24.1
WAP application architecture.

The mobile device, typically a phone, has a piece of resident software known as a *micro-browser*. As the name implies, this piece of software is similar to a Web browser but designed for devices such as mobile phones with limited memory and processing power. Most mobile phones on the market today use OpenWave's (formerly Phone.com) microbrowser. Additionally, the microbrowser is usually designed to render the WML or HDML languages rather than HTML, as described in the next section.

Because the current generation of mobile phones do not inherently know how to communicate with resources on the Internet, the WAP gateway acts as an intermediary between the mobile device and the public Internet. Most WAP gateways are managed by the wireless service provider, and run software created by companies such as OpenWave, Nokia, or SAS.

The target host is generally just a plain old Web server that simply returns content properly formatted for WAP. Proper formatting means that the content is described using WML or, less optimally, by employing a filter to dynamically convert HTML content to WML.

The chief benefit of WAP is its wide support across pretty much all mobile and wireless devices. What's more, the available functionality in WAP is essentially the lowest common denominator of mobile devices, meaning wider compatibility at the expense of powerful functionality. In addition, between the application server, Web server, WAP gateway, microbrowser, and client device, WAP developers have a lot to worry about in their efforts to create applications that function properly for the greatest number of end users.

The chief drawbacks of WAP include the limited screen size and processing capabilities of the devices, the typical lack of a full keyboard for data entry, the slow download speeds, and the fact that wireless airtime for WAP applications can still be expensive.

WML: The Language of WAP

As we mentioned earlier, information is exchanged in WAP using wireless markup language (WML). WML is in some ways modeled after HTML, but WML has two things going for it when compared to HTML. First, it is made up of a relatively small set of tags and attributes, making it compact enough to be used efficiently with machines with little memory and processor muscle. Second, it is based on Extensible Markup Language (XML), so content is well formed and not as open to browser interpretation as is HTML. This chapter is not intended to present a primer on WAP, but we would like to turn you on to some of the basics.

You probably know that HTML is based on a page metaphor, with each `.html` file served to a browser generally representing one page of information. WML, on the other hand, is based on a card deck metaphor, with one `.wml` file representing a deck containing some number of cards. Each card represents one screen of information. In this way, the functionality of an entire WML deck can be sent to a client with only one client-to-server round trip, as opposed to the round-trip-per-page system that is the norm on the Web. A typical `.wml` file, then, might look something like this:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.WAPforum.org/DTD/wml_1.1.xml">
<wml>
  <card>
    <do type="accept">
      <go href="#hello"/>
    </do>
    <p>Punch the Button</p>
  </card>
  <card id="hello">
    <p>Hello from WAP!</p>
  </card>
</wml>
```

If you know just a little about HTML and XML, you can probably figure out this code with relative ease. The document prologue, which makes up the first few lines, is standard XML and describes the XML version of this document and the location of the DTD used to describe the tags and attributes contained within. After that, the code goes on to create a deck with two cards, one with an OK button, and one with a greeting.

WML syntax additionally supports things such as events, timers, field sets, lists, and images (although not all devices support images). Some of the later versions of WAP browsers even support a scripting language called WMLScript. We cannot cover the entirety of the WML language here, but if you're interested, you can view the details of the WML spec at `http://www.WAPforum.org`.

If you want to try your hand at developing some WML content, the easiest way to start is to obtain an emulator. You can obtain the emulator from the microbrowser developer at `http://www.openwave.com`, or two other popular emulators come directly from the mobile communications leaders Nokia and Ericsson; visit `http://forum.nokia.com` or `http://www.ericsson.com/developerszone`. It's always a good idea to get things working on the emulators first before moving on to real hardware because the emulators offer much quicker write-run-debug turnaround time. It's also a good idea to test your final product on as many devices as possible prior to release because each device's unique characteristics can cause your deck to behave or display differently from how you intend.

WAP Security

The WAP specification calls for a wireless encryption stack known as Wireless Transport Layer Security (WTLS) to be used for secure connections. Because SSL is too resource intensive to be used with the current generation of mobile devices, WTLS was created to provide encryption and authentication services between the device and the WAP gateway. The gateway is then able to communicate with Internet hosts via the standard SSL protocol. Despite the fact that both WTLS and SSL are quite secure in themselves, the potential for security breaches exists at the WAP gateway at the point where the WTLS data stream is decrypted and re-encrypted with SSL. WTLS architecture is illustrated in Figure 24.2.

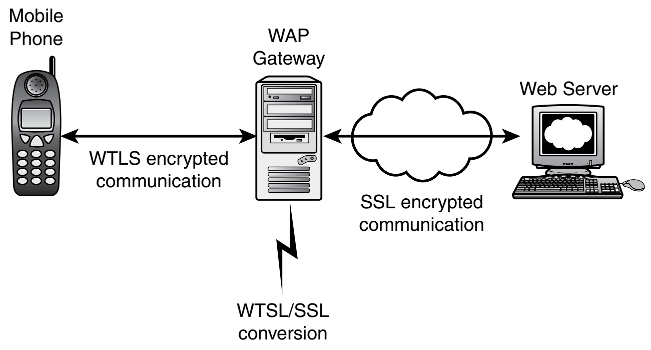
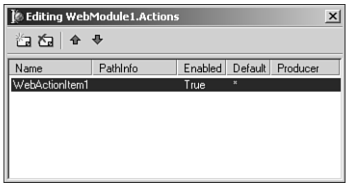


FIGURE 24.2
WAP's Wireless Transport Layer Security.

A Simple WAP Application

Creating a WAP application in Delphi is little different from creating a regular Web application in Delphi. WAP is perhaps even easier to target because the limitations inherent in WAP and the target devices tend to beget simpler applications on the server side than traditional browser-based applications. The opposite side of this coin, however, is that it is more challenging for developers to develop applications that are engaging to useful to end users given these limitations.

For this example, start by creating a normal WebBroker application as you learned in Chapter 23, "Building WebSnap Applications." This application has a single Web module with a single action. This action is marked as default as shown in Figure 24.3.



Name	PathInfo	Enabled	Default	Producer
WebActionItem1		True	*	

FIGURE 24.3
A simple WebBroker WAP application.

Listing 24.1 shows the source code for the main unit of this application, including the `OnAction` event handler for the Web module's default action.

LISTING 24.1 `Main.pas`—The Main Unit for the `SimpWap` Project

```
unit Main;

interface

uses
  SysUtils, Classes, HTTPApp;

type
  TWebModule1 = class(TWebModule)
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

LISTING 24.1 Continued

```
var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}

const
  SWMLContent = 'text/vnd.wap.wml';
  SWMLDeck =
    '<?xml version="1.0"?>'#13#10 +
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"'#13#10 +
    '"http://www.WAPforum.org/DTD/wml_1.1.xml">'#13#10 +
    '<wml>'#13#10 +
    '  <card>'#13#10 +
    '    <do type="accept">'#13#10 +
    '      <go href="#hello"/>'#13#10 +
    '    </do>'#13#10 +
    '    <p>Punch the Button</p>'#13#10 +
    '  </card>'#13#10 +
    '  <card id="hello">'#13#10 +
    '    <p>Hello from WAP!</p>'#13#10 +
    '  </card>'#13#10 +
    '</wml>'#13#10;


procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.ContentType := SWMLContent;
  Response.Content := SWMLDeck;
end;

end.
```

When the action is invoked, the event handler responds by setting the `ContentType` and `Content` properties of the `Response` object. `ContentType` is set to the WML content type string, and the content returned is the same simple WAP deck that was explained earlier in this chapter.

NOTE

Remember to set the `ContentType` of the `Response` object to the string containing the MIME type of the variety of content you are returning. This sets the content type information in the HTTP header. If you return the incorrect content type, your content will likely be misinterpreted on the target device. Some notable WAP content types include

- `text/vnd.wap.wml` for WML code
- `text/vnd.wap.wmlscript` for WML script code
- `image/vnd.wap.wbmp` for wireless bitmap images

Figure 24.4 shows this simple Delphi WAP application in action.

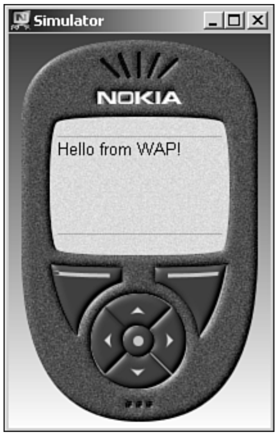


FIGURE 24.4
Delphi WAP application in action.

Error Reporting

The default exception handler for WebSnap applications sends an HTML message to the client with information on the error. Of course, most WAP devices will not be able to understand an HTML error, so it's important to ensure that any errors that might occur in your WAP application are surfaced as WML messages to the client rather than HTML. This can be done by wrapping each `OnAction` event handler with a `try..except` block that calls out to an error message formatting routine. This is shown in Listing 24.2.

Wireless Bitmaps

Although WAP doesn't yet support the fancy JPEG and GIF graphics common on the Web, most WAP devices support monochrome images in the form of wireless bitmaps (wbmp). Listing 24.2 adds a new action to the Web module to support the generation of a wbmp. This action generates an official-looking but quite random graph for display on the target device. Although we won't delve into the binary format of wbmp files in this text, you can see that it isn't a great deal of work to generate wbmps manually in your WAP applications. Figure 24.5 shows what a WBMP will look like on a phone display.

NOTE

Not all WAP browsers, devices, and emulators support wbmp images. Be sure to test before assuming support.

LISTING 24.2 `Main.pas`—Once More with Feeling

```
unit Main;

interface

uses
  SysUtils, Classes, HTTPApp;

type
  TWebModule1 = class(TWebModule)
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
    procedure WebModule1GraphActionAction(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    procedure CreateWirelessBitmap(MemStrm: TMemoryStream);
    procedure HandleException(e: Exception; Response: TWebResponse);
  end;

var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}
```

LISTING 24.2 Continued

```
const
  SWMLContent = 'text/vnd.wap.wml';
  SWBMPContent = 'image/vnd.wap.wbmp';
  SWMLDeck =
    '<?xml version="1.0"?>'#13#10 +
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"'#13#10 +
    '"http://www.WAPforum.org/DTD/wml_1.1.xml">'#13#10 +
    '<wml>'#13#10 +
    '  <card>'#13#10 +
    '    <do type="accept">'#13#10 +
    '      <go href="#hello"/>'#13#10 +
    '    </do>'#13#10 +
    '    <p>Punch the Button</p>'#13#10 +
    '  </card>'#13#10 +
    '  <card id="hello">'#13#10 +
    '    <p>Hello from WAP!</p>'#13#10 +
    '  </card>'#13#10 +
    '</wml>'#13#10;

  SWMLError =
    '<?xml version="1.0"?>'#13#10 +
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"'#13#10 +
    '"http://www.wapforum.org/DTD/wml_1.1.xml">'#13#10 +
    '<wml>'#13#10 +
     '  <card id="error" title="SimpWAP">'#13#10 +
     '    <p>Error: %s'#13#10 +
     '      <do type="prev" label="Back">'#13#10 +
     '        <prev/>'#13#10 +
     '      </do>'#13#10 +
     '    </p>'#13#10 +
     '  </card>'#13#10 +
     '</wml>'#13#10;



procedure TWebModule1.HandleException(e: Exception; Response: TWebResponse);
begin
  Response.ContentType := SWMLContent;
  Response.Content := Format(SWMLError, [e.Message]);
end;

procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
```

LISTING 24.2 Continued

```
begin
  try
    Response.ContentType := SWMLContent;
    Response.Content := SWMLDeck;
  except
    on e: Exception do
      HandleException(e, Response);
  end;
end;

procedure TWebModule1.WebModule1GraphActionAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  MemStream: TMemoryStream;
begin
  try
    MemStream := TMemoryStream.Create;
    try
      CreateWirelessBitmap(MemStream);
      MemStream.Position := 0;
      with Response do
      begin
        ContentType := SWBMPContent;
        ContentStream := MemStream;
        SendResponse;
      end;
    finally
      MemStream.Free;
    end;
  except
    on e: Exception do
      HandleException(e, Response);
  end;
end;

procedure TWebModule1.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, Dir, Bit: Integer;
  B: Byte;
```

LISTING 24.2 Continued

```
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 }
  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;

initialization
  Randomize;
end.
```

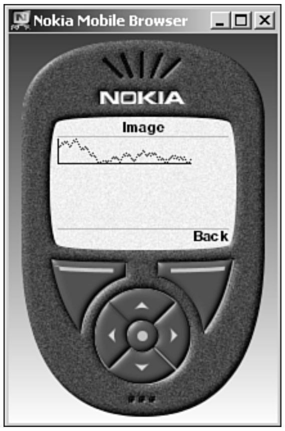


FIGURE 24.5
Viewing the WBMP in the Nokia emulator.

I-mode

I-mode is a proprietary technology for Internet content on mobile phones developed by NTT DoCoMo, Japan's telecommunications behemoth. I-mode is very successful in Japan, with over 20 million subscribers and growing. In many ways, i-mode is everything WAP isn't: It supports rich 256-color graphics, color phone displays, and uses an "always-on" TCP/IP connection. Additionally, DoCoMo has developed a revenue sharing model that enables i-mode sites to get a slice of the financial pie based on usage. However, i-mode is hampered by the "P" word (proprietary) and availability outside Japan is scarce; i-mode services will be rolling out in the United States, United Kingdom, and continental Europe beginning this year.

From a developer's standpoint, targeting i-mode phones isn't much more difficult than generating content for the Web because i-mode content is developed using a subset of HTML known as Compact HTML (cHTML). Supported cHTML tags and rules are available from DoCoMo at `http://www.nttdocomo.com/i/tagindex.html`. Note that in addition to using only cHTML-supported tags, i-mode sites must also ensure that the S-JIS character set is used, images are in GIF format, and pages have no script or Java content.

PQA

Palm Query Applications (PQA) are essentially normal HTML pages stripped of add-ons such as scripting and images that are designed for display on the screen of a wireless PalmOS device. Wireless PalmOS devices, such as Palm VIIx devices or those equipped with Novatel modems, are currently limited to North America. Like i-mode, PQAs are developed using a subset of HTML, except Palm has added a few proprietary extensions. Developers interested in creating PQAs should download the Web Clipping Developer's Guide from Palm at `http://www.palmos.com/dev/tech/docs/`.

In general, the PQA flavor of HTML includes everything in HTML 3.2 with the exception of applets, JavaScript, nested tables, image maps, and the `VSPACE`, `SUB`, `SUP`, `LINK`, and `ISINDEX` tags. PQAs also include several interesting additions to HTML. Most notable among these are the `palmcomputingplatform` meta tag and `%zipcode` and `%deviceid` tags. When an HTML document contains the `palmcomputingplatform` meta tag, this serves as an indicator to Palm Computing's Palm.net proxy (which acts as the intermediary between a Web server and Palm device, not unlike a WAP gateway) that the document is optimized for display on a PalmOS handheld and doesn't require parsing to be stripped of invalid content. When the `%zipcode` tag appears in a requested posted by the client, the Palm.net proxy will replace the tag with the ZIP Code where the device is located (based on radio tower information). The `%deviceid` tag similarly sends the PalmOS's device's unique ID to the server. This is particularly handy because PQA HTML does not support cookies, but a similar means of state management can be crafted using the `%deviceid` tag.

With Web clipping, Palm has taken a different approach than most other players in this space. Rather than use a browser-like entity to navigate to a site and pull down content, PQAs exist locally on the PalmOS device. PQAs are built by running a standard .HTML file through a special compiler that links the HTML with referenced graphic files and other dependencies. Users install PQAs like normal PalmOS PRC applications. PQAs gain efficiency by including portions of the application local and only going out to the network for "results" pages.

PQA Client

The first step toward developing a PQA application is to create the piece that will physically reside on the client device. This is done by creating an HTML document and compiling it using Palm Computing's PQA Builder tool. A sample PQA HTML document is shown in Listing 24.3.

LISTING 24.3 An HTML Document for a PQA

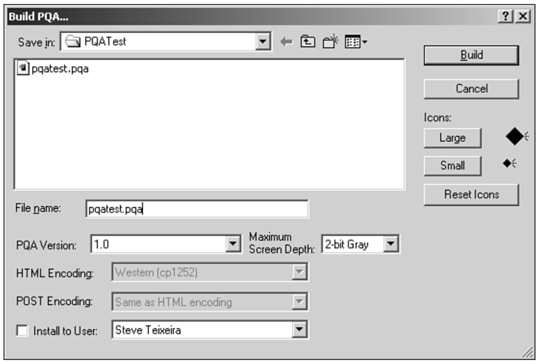
```
<html>
<head>
<title>DDG PQA Test</title>
<meta name="palmcomputingplatform" content="true">
</head>
<body>
<p>This is a sample PQA for DDG</p>
<img src="image.gif">
<form method="post" action="http://128.64.162.164/scripts/pqatest.dll">
<input type="hidden" value="%zipcode" name="zip">
```

LISTING 24.3 Continued

```
<input type="hidden" value="%deviceid" name="id">
<input type="submit">
</form>
</body>
```

You can see that this simple HTML document contains a reference to an image, some text, a form with a submit button, and hidden fields used to pass the ZIP Code and device ID to the server.

Figure 24.6 shows this document being compiled in PQA Builder.



Save in: PQATest

pqatest.pqa

File name: pqatest.pqa

PQA Version: 1.0

Maximum Screen Depth: 2-bit Gray

HTML Encoding: Western (cp1252)

POST Encoding: Same as HTML encoding

Install to User: Steve Teixeira

FIGURE 24.6
Compiling with PQA Builder.

Once compiled, a file with a `.pqa` extension is generated. This file contains the HTML document as well as any referenced images. This file can be installed onto the PalmOS device the same as any other PalmOS application.

PQA Server

The server-side portion, like WAP, is a WebSnap application that handles the page requests from clients and returns pages. Unlike WAP with its WML, however, PQAs communicate using the HTML variant described previously. Listing 24.4 shows the main unit of a WebBroker application designed to fulfill the server role for the PQA client described previously.

LISTING 24.4 `Main.pas`—the Main Unit for the `PQATest` Application

```
unit Main;

interface
```

Listing 24.4 Continued

```
uses
  SysUtils, Classes, HTTPApp;

type
  TWebModule1 = class(TWebModule)
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}

const
  SPQAResp =
    '<html><head><meta name="palmcomputingplatform" content="true"></head>'+
    #13#10 +
    '<body>Hello from a Delphi server<br>Your zipcode is: %s<br>'#13#10 +
    'Your device ID is: %s<br><img src="file:pqatest.pqa/image.gif"></body>'+
    '</html>';

procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content := Format(SPQAResp, [Request.ContentFields.Values['zip'],
    Request.ContentFields.Values['id']]);
end;

end.
```

The server responds to the client by sending the client's ZIP Code and device ID. An interesting technique in the HTML code returned by the server is that it references the same image that was compiled into the `.pqa` on the client side using the `file:<pqaname>` syntax. This allows you to build rich graphics into your PQAs by compiling them into the client side and referencing them on the server, thereby obviating the need to download any graphics over the wireless modem. Figures 24.7 and 24.8 show this application in action, before and after the submit button is pressed. Note that the ZIP Code and device ID are `null` values in the emulator.

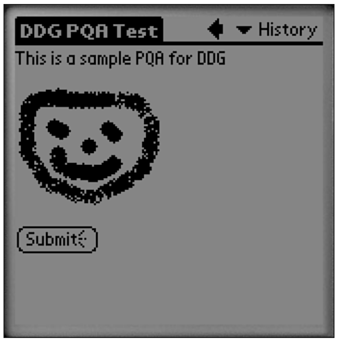
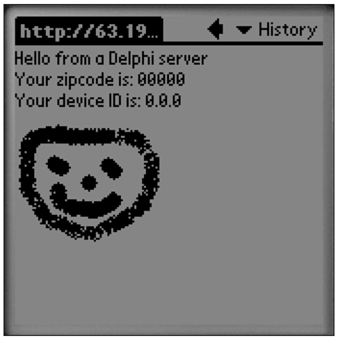


FIGURE 24.7
PQATest in the PalmOS emulator.



Hello from a Delphi server
Your zipcode is: 00000
Your device ID is: 0.0.0

FIGURE 24.8
PQATest in the PalmOS emulator after pressing Submit.

Wireless User Experience

User experience is by far the most important factor in determining whether a mobile system will ultimately prove useful to individuals. However, user experience is all-too-often given short shrift in favor of gratuitous features or technology. Because of inherent limitations in the mobile world—particularly connectivity and device size—there is little room for error in a developer's attempt to provide users with the functionality they need when they need it. The trick is to focus on the user: Determine what information or service the users need and endeavor to get it to them as efficiently as possible.

Circuit-Switched Versus Packet-Switched Networks

When considering the mobile phone as a client platform, one issue that can have a dramatic impact on usability is whether the mobile phone network is circuit-switched or packet-switched. Circuit-switched networks operate like a modem on a conventional phone: you must dial-in to establish a direct connection with the host, and that connection must be maintained while data exchange is taking place. Packet-switched networks behave more like a fixed Ethernet connection: The connection is always active, and data packets can be sent and received out over the connection at any time.

The overhead required to establish connections in circuit-switched networks can often be a major impediment to user satisfaction—particularly if the user is performing a task that should only take a few seconds. After all, who wants to wait up to 20 seconds to connect in order to interact with an application for 3 seconds? Most mobile networks today are still circuit-switched, but notable example of networks utilizing packet-switched technology include NTT DoCoMo's i-mode, Nextel's iDEN, and AT&T's CDPD networks.

Wireless Is Not the Web

A common misconception among purveyors of mobile devices and wireless networks is that people desire to surf the Web on these devices. With their tiny screens and limited bandwidth, mobile devices are an exceedingly inconvenient vehicle for Web surfing. Each page of information can take several seconds to fetch due to network constraints, and entering data can be downright painful—particularly on a mobile phone. Instead, mobile applications need to be optimized to deliver specific information and services with minimal data entry required on the part of the user and as few client-to-server roundtrips as feasible.

The Importance of Form Factor

When designing mobile applications, you must always remain sensitive to the available amount of screen real estate. Whether creating a WAP-based application bound to a microbrowser or a custom user interface in J2ME, application developers have to balance the issues of communicating a sufficient quantity of information in each screen against maintaining a readable and navigable user interface.

Data Entry and Navigation Techniques

Related to form factor is the issue of data entry. Different types of devices rely on different mechanisms for data entry. PalmOS and Pocket PC devices use a combination of a stylus and handwriting recognition (with optional portable keyboards); RIM BlackBerry devices use thumb-size keyboards; mobile phones generally have only a numeric keypad and a few extra buttons. This means that applications designed for one mobile platform might be difficult to use on another. For example, a PDA stylus is great for tapping random areas of the screen, whereas a BlackBerry is better suited toward text data entry, and a phone is best suited for as little data entry as possible!

M-Commerce

Just as e-commerce has come to refer to commercial transactions performed over the Web, *m-commerce* refers to commercial transactions over mobile devices. Developers of mobile commerce sites must understand that m-commerce is fundamentally different from e-commerce. Most conspicuously, it's not feasible for mobile customers to browse for items. In the case of a mobile phone, for instance, it's too time-consuming to enter keystrokes—images either don't exist or are of poor quality, and there isn't enough screen real estate to describe items.

Instead, m-commerce systems should be designed with the notion in mind that the users know what they want; just make it easy for them to give you their money. Remember: if the user wants to buy a television or book, there's little reason why he wouldn't simply wait until he get to his home or office to make the purchase on a full-sized computer. The fact that the user is even willing to engage in m-commerce implies that there is some sense of immediacy or urgency in making the purchase, and winning m-commerce merchants will be the ones who recognize and take advantage of this fact.

For example, one eastern European country allows motorists to pay parking meter tolls using their mobile phone. This seems like a relatively simple application on the surface, but the value proposition for both parties is very compelling. The motorist doesn't have to worry about whether he has enough coins to feed the meters, and the meter operator isn't burdened with collecting coins from dozens or hundreds or thousands of meters and trucking the coins to the bank. Police monitoring the meters can use a mobile device tied to the same system to know at an instance how much time is left for a given space.

Summary

The world of mobile computing has grown dramatically in recent years, and it can be difficult to keep track of emerging trends. Our hope is that at this point, you are now armed with enough information to make some strategic decisions and move forward with a mobility project. In addition, you have seen that Delphi is a very capable tool when it comes to building this next generation of wireless applications.