Developing Software for the QUALCOMM BREW Platform

RAY RISCHPATER

Apress™

Developing Software for the QUALCOMM BREW Platform

ISBN (pbk): 1-59059-116-X

Printed and bound in the United States of America 12345678910

Technical Reviewer: Shane Conder

Editorial Board: Dan Appleman, Craig Berry, Gary Cornell, Tony Davis, Steven Rycroft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Nate McFadden

Copy Editor: Kim Wimpsett

Production Manager: Kari Brooks

Production Editor: Janet Vail

Proofreader: Kim Cofer

Compositor: Argosy Publishing

Indexer: Carol Burbo

Artist: Cara Brunk

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States, phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section.

CHAPTER 1

Introducing the QUALCOMM BREW Platform

THE QUALCOMM BINARY Runtime Environment for Wireless (BREW) platform is an exciting step forward in mobile application development. Within the QUALCOMM BREW platform there's a powerful set of Application Programming Interfaces (APIs) that you use with C or C++ to build your application for wireless handsets. Once you complete your application, you can submit it to QUALCOMM for external validation and from there to wireless carriers around the world for your customers to purchase and enjoy.

This chapter gives you a quick overview of the QUALCOMM BREW platform, including what it can and can't do and why carriers, handset manufacturers, and developers have been keen to adopt the platform. After examining the platform from a marketing perspective, the chapter sketches the architecture of the QUALCOMM BREW application, as well as gives you a thumbnail view of how to organize a QUALCOMM BREW application. Finally, the chapter closes with your first sample application written for QUALCOMM BREW—the now-famous HelloWorld program.

Seeing the QUALCOMM BREW Platform for the First Time

If you've never used a QUALCOMM BREW–enabled handset, the first thing you should do is go get one and play with it. Books such as this one can't fully capture the experience of using a QUALCOMM BREW–enabled handset, but it'll try.

Most QUALCOMM BREW–enabled handsets look like regular cell phones and have the following features in common:

- A multiline color, grayscale, or monochrome display used by the handset for both telephony operations and QUALCOMM BREW applications
- A directional keypad or arrow keys to navigate a selection cursor up, down, left, and right
- A selection key

- Alphanumeric entry support through either multitap (where you press a number key one or more times to navigate through successive letters) or other mechanisms such as Tegic's T9 predictive keyboard

- A means to launch QUALCOMM BREW applications, either via a configuration menu or a dedicated key on the handset

In addition, most newer QUALCOMM BREW–enabled handsets support the third-generation wireless networks (often called *express networks*) and on-handset global positioning for emergency response and location applications.

What differentiates these handsets from most other programmable handsets—including those running the Palm Powered and Microsoft Pocket PC for Smartphone platforms—is that they are priced to be competitive with today's wireless handsets rather than with handheld computers. This helps ensure the rapid adoption of QUALCOMM BREW–enabled handsets by consumers—even consumers who may not immediately want to run an application on their handset.

You can launch a QUALCOMM BREW application (called an *applet*) using a phone control to bring up the application menu, which looks something like Figure 1-1.

NOTE *Both the mechanism for launching the QUALCOMM BREW application menu and the appearance of the BREW application menu can differ from handset to handset. Many QUALCOMM BREW–enabled handsets offer a button labeled with a carrier-dependent logo, such as the Verizon Wireless Get It Now logo, and others use a menu option or sequence of keys. Similarly, the appearance of the QUALCOMM BREW application menu may differ slightly from what you see in Figure 1-1.*

Figure 1-1. The QUALCOMM BREW application menu in action

You can obtain new QUALCOMM BREW–enabled applications by using the Mobile Shop application, which wirelessly obtains a list of the latest applications and lets you purchase them for immediate use on the phone. When you launch an application such as RocketMileage, the application consumes the entire screen (as shown in Figure 1-2), and you can freely interact with the application. If you receive a telephone call or Short Message Service (SMS) message while you're using the application, the handset pauses your application and shows a dialog box that lets you choose to accept the call or message or continue using the application.

Figure 1-2. The RocketMileage application

Choosing the QUALCOMM BREW Platform

Making the decision to develop applications for the QUALCOMM BREW platform hinges on three factors: the wireless carriers your application targets, the market your application targets, and the flexibility QUALCOMM BREW provides you with as a developer. Equally important, you should understand why you might *not* want to use QUALCOMM BREW to develop your application. The following sections take a closer look at each of these issues.

Why Carriers Choose QUALCOMM BREW

Wireless carriers face two challenges directly affecting their profits: decreasing Average Revenue Per User (ARPU) and increasing subscriber demand. Competition between wireless carriers (and legislation in some countries) continues driving down what carriers can charge their customers, even as carriers must continue to invest to upgrade their infrastructures to support increasing demand. Paradoxically, however, the increased revenue from new customers doesn't fully offset the costs carriers incur when upgrading their networks to handle the additional customer demand.

Consequently, carriers are in the awkward position of needing to drive additional demand for existing services in the hopes of raising ARPU and increasing profits. By opening the wireless network to third-party applications, carriers want software developers to provide value-added applications that increase subscriber network use and raise ARPU.

Initial attempts to spur demand by offering third-party applications have met with limited success for several reasons. First, few handsets are available for any one carrier that can run third-party applications. These handsets, often called *smart phones* or *wireless terminals*, are a hybrid handheld computer (often running Palm OS, Windows Pocket PC, or Symbian OS) and wireless handset that typically costs several times more than a phone. Second, these smart phones are difficult to use, often with limitations on battery life and with user interfaces more similar to computers than phones. Third, installing third-party applications on these devices is an arcane process generally requiring a computer and separate cable (even the first generation of Java-enabled handsets suffer this drawback) so that out of the small fraction of subscribers that *use* one of these handsets, even fewer actually *use* them more than they use a traditional wireless handset.

QUALCOMM BREW has features that meet each of these drawbacks. First, QUALCOMM BREW is lightweight; it's essentially a thin veneer over a telephone's native operating system, and it often provides interfaces to hardware features of today's handsets (such as media decoding) that are implemented in silicon on the device. Consequently, QUALCOMM BREW is available on even the least expensive wireless handsets, making it easy for carriers to offer QUALCOMM BREW to subscribers with little or no added cost. Second, both QUALCOMM BREW itself and the handsets on which it's available are primarily wireless handsets. These devices bear little resemblance to traditional computing devices, making them comfortable to nearly all users. Finally, QUALCOMM BREW provides an end-to-end software distribution platform for carriers and consumers. As you'll see in the next section, you have access to third-party applications when using a QUALCOMM BREW–enabled handset. When you purchase an application, the application is wirelessly transferred to your phone, and the cost of the application is added to your phone bill. The QUALCOMM BREW platform handles the necessary billing, maintains a secure transaction between handset and server, and even provides developers with royalty checks from carriers on a regular basis from application sales.

Why Developers Choose QUALCOMM BREW

So why should *you* choose QUALCOMM BREW? If you're looking to develop and deploy applications for wireless handsets, QUALCOMM BREW provides several advantages.

First, QUALCOMM BREW is likely the most widely distributed platform for handset software development in the world. Within the first year of QUALCOMM BREW's introduction to the market, QUALCOMM BREW was in the hands of more than a million users. Moreover, it's seeing rapid and widespread adoption in the United States and abroad, often replacing older smart phones in the process. This is largely because when carriers offer QUALCOMM BREW handsets, they generally do so across all price points, making QUALCOMM BREW available on all handsets. Consequently, the applications you write are likely to be available to all subscribers for a wireless carrier as soon as a carrier adopts QUALCOMM BREW.

Second, QUALCOMM BREW uses concepts and programming languages familiar to the experienced software developer. You write your applications in C or C++ using the BREW APIs, test them in a simulator on your desktop computer, and download them directly to the handset for testing. Although this is similar to development for other platforms including Sun's Java 2 Micro Edition (J2ME) for handsets and the Palm Powered platform for handsets, there are fewer hurdles because the APIs and tool chain remain largely unchanged for QUALCOMM BREW.

Third, QUALCOMM BREW is truly a write-once, run-anywhere platform because the APIs have been established by QUALCOMM and are available on *all* platforms. In other words, it's not a community process with different manufacturers providing various extensions, as is common with J2ME, the Palm Powered platform, and Microsoft Pocket PC Smartphone edition. Most of the QUALCOMM BREW APIs are available on all handsets, and few handset vendors extend the QUALCOMM BREW APIs in unconventional ways.

Why Developers Choose Other Platforms

Before closing this section, you should understand why you might *not* want to choose QUALCOMM BREW for your application. Obviously, if your customers use a cellular network that doesn't support QUALCOMM BREW, it makes little sense to write your applications using QUALCOMM BREW. If you're porting business logic from another application—say, one in Java or one running the Palm Powered platform for a handheld computer—it may make sense for you to use a super phone running the same platform, rather than trying to port your entire application to QUALCOMM BREW. This is especially true if you're writing a *vertical application* (an application targeted to a particular market), where you can control your customers' hardware configuration and require a specific wireless carrier and wireless handset.

NOTE *At the time of this book's publication, vendors including Insignia and IBM have demonstrated J2ME Mobile Information Device Profile (MIDP) runtimes for the QUALCOMM BREW platform. By the time you read this, it's quite likely that the J2ME MIDP will be available for QUALCOMM BREW on at least some handsets and you'll be able to run your J2ME MIDP applications on QUALCOMM BREW–enabled handsets with no difficulty. If this comes to pass, you'll have yet another alternative for deploying applications on QUALCOMM BREW: either native applications in C or C++ using the QUALCOMM BREW APIs or traditional J2ME MIDP midlets running atop a third-party Java virtual machine that uses QUALCOMM BREW.*

Understanding the QUALCOMM BREW Platform

When you first open the QUALCOMM BREW Software Development Kit (SDK), you may be daunted by all of the new terms, APIs, and conventions. Although it's easy to get confused—especially if you're the type who dives into something without looking at the documentation—it's not as difficult as it looks to grasp the fundamentals.

Confusion usually centers around three aspects of QUALCOMM BREW: understanding the relationship between modules, applications, and classes; understanding Module Information File (MIF) and resource files; and understanding the application delivery process. Learning about each of these topics now makes it far easier for you to understand other aspects of the QUALCOMM BREW platform later.

Understanding the Relationship Between Modules, Applications, and Classes

A *module* is a chunk of executable code in QUALCOMM BREW, much like a shared library on other platforms. Modules contain definitions of classes, or the implementations of interfaces you use to build applications. Many of these classes are loaded from the handset's Read-Only Memory (ROM) on demand when your application needs to use them; however, you can define your own classes to use in your application. When you do so, you can also choose to share these classes with other developers; in this case, the module is called an *extension* because the handset obtains it online when the classes are required.

As on most other object-oriented platforms, your *applet* (QUALCOMM BREW's name for applications) itself is a class—specifically, a subclass of the

IApplet class. It must implement the interface defined by the IApplet class to process events the system sends in response to system and user interface events. Note, too, that because an application is simply a class, a module can contain more than one applet.

Every class—be it a system class, one of your classes, or an applet—must have a unique class identifier (or *class ID*). A class ID is a 32-bit integer and is allocated by QUALCOMM as a service to registered developers at the QUALCOMM BREW extranet (available from `http://www.qualcomm.com/brew/`). When obtaining an interface to a class, you must use its class ID to request the interface from the system shell, the only interface loaded when your applet first runs. (An exception to this is the `IShell`, `IModule`, and `IDisplay` interfaces; your application will receive instances of these interfaces when it executes.)

NOTE *When you create your unique class ID using the extranet, the extranet will offer to let you download a BREW ID (BID) file, which contains a C-style definition of your class ID. You'll need to use this BID file when creating your application, so be sure to set it aside in your project directory.*

You build modules in two ways when developing your applet. During most of your development, you use Microsoft Visual Studio to build Dynamically Linked Libraries (DLLs) you invoke through the QUALCOMM BREW handset simulator. Periodically you build your modules using the ARM compiler and QUALCOMM tool chain, resulting in module files (which end in .mod and are sometimes called *MOD files*) that you transfer to your handset for on-device testing.

Understanding MIF and Resource Files

In addition to your applet's module file, your applet needs an MIF to describe the applet to the handset's runtime. As the handset starts up, it reads each applet's MIF to determine the applet's name, icons, and class ID. The MIF also contains additional information, such as the author of the application and a set of flags delineating various kinds of behaviors the application performs (such as file system access or network transactions). You build MIFs using the BREW MIF Editor, a component of the QUALCOMM BREW SDK.

In many of your applets, you want to include strings, icons, and dialog boxes. You can include these items in your application using the BREW Resource Editor, which lets you add items to your application. Once you use the BREW Resource

Editor to add items to your application, it will create a C header file and a resource file (called a *bar* file, short for *BREW Application Resource* file) for your application.

Chapter 3, "Developing for the QUALCOMM BREW Platform," discusses both of these tools in greater detail.

Understanding the Application Delivery Process

Deploying your QUALCOMM BREW application is quite different from deploying traditional shrink-wrapped applications for existing desktop and handheld platforms. Because your application will run on wireless handsets that are expected to never fail, your application must meet a high standard of quality. Moreover, the deployment process must be different to accommodate the over-the-air application download features of QUALCOMM BREW.

To meet these goals, you must submit your completed application to QUALCOMM for validation. To do this, you must first become a registered QUALCOMM BREW developer (discussed fully in the "Becoming an Authenticated QUALCOMM BREW Developer" section). Next, you need to use the QUALCOMM BREW tools to cryptographically sign a snapshot of the module, resource file, and MIF. You electronically submit these, along with your user manual and test plan, to QUALCOMM for verification.

In turn, QUALCOMM validates your application and provides you with the validation results. If your application passes the testing, QUALCOMM assigns your application a unique part number and certifies it as a *TRUE BREW* application. You may then offer your application—by referring to its part number, not by submitting new binaries!—to wireless carriers such as Verizon Wireless to offer to its customers.

As customers purchase your application, your carrier's BREW Mobile Shop tracks customer purchases, charging customers' regular wireless service bills accordingly. Quarterly you receive payments aggregated by QUALCOMM for your application sales, and you can check your sales by part number using the QUALCOMM extranet at any time.

Creating Your First Application

By now, you're probably wondering exactly what a QUALCOMM BREW application looks like from the inside out or at least how you can get started writing applications for QUALCOMM BREW. So let's stop the chatter about what QUALCOMM BREW is about and see a QUALCOMM BREW application in action!

Becoming an Authenticated QUALCOMM BREW Developer

Although it's not imperative that you do so immediately, it's helpful to become an authenticated QUALCOMM BREW developer as soon as you start developing for QUALCOMM BREW. By doing so, you receive a number of advantages, including the ability to run applications on test handsets and obtain unique class IDs for your applications.

To become an authenticated developer, visit `http://www.qualcomm.com/brew/` and follow the links through the developer support section to the Become an Authenticated Developer page. You'll need to apply for a VeriSign certificate, complete the BREW Independent Solutions Vendor (ISV) development form, and register with National Software Testing Labs (NSTL), which is the validation firm QUALCOMM uses to test your applications prior to giving them TRUE BREW certification.

Installing the QUALCOMM BREW SDK

Although you don't need to be authenticated to begin QUALCOMM BREW development, you *do* need the QUALCOMM BREW SDK. It's available from the QUALCOMM BREW Web site at `http://www.qualcomm.com/brew/`. You need to install Microsoft Visual Studio first and then download and install the QUALCOMM BREW SDK.

Installing the SDK is easy; it's packaged as an installer for Microsoft Windows. It includes the headers and libraries you'll need as well as the documentation, sample applications, and the BREW Emulator for Microsoft Windows, which you need to test your application on your desktop computer.

NOTE *Throughout this book, I refer to the BREW 2.0 SDK, including the BREW 2.0 SDK tool chain, even though many of the examples operate correctly on QUALCOMM BREW 1.0 and 1.1. Where a specific version of QUALCOMM BREW is required, I note it in the text that accompanies the example.*

If you've become an authenticated QUALCOMM BREW developer, you should also go to the QUALCOMM BREW extranet (available from `http://www.qualcomm.com/brew/`) and download the tools available to authenticated developers. These tools include the applications you need to install, test, and execute your application on a wireless handset.

Writing Your First Application

With the tools installed, it's time to write your first application. Let's start with the simple HelloWorld application that launches and draws the words *Hello World* on the display.

CAUTION *Although you can build applications with Microsoft Visual Studio .NET or Microsoft Visual Studio 6.0, the following instructions are for Microsoft Developer Studio 6.0. Be sure to check the documentation included with Microsoft Visual Studio .NET for further details.*

To create the HelloWorld program, follow these steps:

1. Go to the QUALCOMM BREW developer extranet and create a class ID named *HELLOWORLD* for your application. If you haven't created a developer account yet, you can borrow the class ID from the QUALCOMM HelloWorld example by copying the file helloworld.bid to your project directory and naming it *hello.bid*.

2. Use Microsoft Paint or another paint application to create an icon for your application. Icons should be no more than 26-by-26 pixels and should be saved as uncompressed bitmaps (Microsoft Windows BMP). Be sure that the icons you create are black and white for black-and-white phones and 8-bit color for color phones. Of course, you can also borrow one of the sample icons from the QUALCOMM BREW SDK during this step. In either case, save the icon in your project directory.

3. Launch Microsoft Visual Studio.

4. Launch the BREW Wizard by selecting File ➤ New ➤ Projects ➤ BREW Application Wizard.

5. Choose a destination for your project and name it *hello*.

6. Click OK.

7. Because your application only draws to the screen, leave all of the options under What Support Would You Like to Include? unchecked and click Next.

8. Launch the MIF Editor by clicking the MIF Editor option to create an MIF for your application.

9. Within the MIF Editor, select the BID file you created in the first step by clicking Browse for BID File.

10. By Name, enter *Hello* as the application name.

11. By Applet Type, select Tools.

12. Select the icon you created in the second step by clicking Browse.

13. Save the MIF using File ➤ Save As and naming it *hello.mif*. (If you're only going to run this in the emulator, you don't need to worry about selecting an MIF type option.)

14. Exit the MIF Editor using File ➤ Exit.

15. Return to Microsoft Visual Studio and click Finish. Then click OK.

16. Under the File View, open the Hello.c file and replace its contents with the contents of Listing 1-1.

NOTE *The source code for this applet (and all of the sample applications from this book) is available at the Apress Web site (*`http://www.apress.com/`*) in the Downloads section, so you can simply use Microsoft Visual Studio to load the Hello.dsw project.*

Listing 1-1. The HelloWorld Applet

```
 1: /**
 2:  *  @name Hello.c
 3:  *
 4:  *  @author Ray Rischpater
 5:  *  Copyright (c) 2001 - 2002 Ray Rischpater.
 6:  *  Portions copyright (c) 2001 - 2002 QUALCOMM, Inc.
 7:  *  @doc
 8:  *  A sample application that draws Hello World
 9:  */
10: #include "AEEModGen.h" // Module interface definitions
11: #include "AEEAppGen.h" // Applet interface definitions
12: #include "AEEDisp.h"   // Display interface definitions
```

```
13: #include "hello.bid"   // Applet class ID
14:
15:
16: /*
17:  * Private function prototypes
18:  */
19: static boolean HelloWorld_HandleEvent( IApplet * pi,
20:                       AEEEvent eCode,
21:                       uint16 wParam,
22:                       uint32 dwParam );
23:
24: /**
25:  * Create an instance of this class. This constructor is
26:  * invoked by the BREW shell when the applet is launched.
27:  *
28:  * @param AEECLSID clsID: class ID of the class being requested
29:  * @param IShell *pIShell: a pointer to the BREW shell
30:  * @param IModule *po: a pointer to the current module
31:  * @param void **ppObj: a pointer to the created applet
32:  * @return AEE_SUCCESS on success, with the applet in *pobj.
33:  */
34: int AEEClsCreateInstance( AEECLSID clsID,
35:               IShell * pIShell,
36:               IModule * po,
37:               void ** ppObj )
38: {
39:   boolean result;
40:   *ppObj = NULL;
41:
42:   // If it's this class being requested...
43:   if( clsID == AEECLSID_HELLOWORLD)
44:   {
45:     // Use the BREW helper function to
46:     // create an instance of this class
47:     result= AEEApplet_New( sizeof(AEEApplet),
48:                            clsID,
49:                            pIShell,
50:                            po,
51:                            (IApplet**)ppObj,
52:                            (AEEHANDLER)HelloWorld_HandleEvent,
53:                            NULL);
54:   }
55:   return result ? AEE_SUCCESS : EFAILED;
56: }
57:
```

```
/**
 * Handles incoming events from the shell.
 *
 * @param IApplet *pi: pointer to this applet.
 * @param AEEEvent eCode: event to handle
 * @param int wParam: word argument associated with event
 * @param uint32 dwParam: double word arg associated with event
 * @return TRUE if the event was handled, FALSE otherwise.
 */
static boolean HelloWorld_HandleEvent( IApplet *pi,
                      AEEEvent eCode,
                      uint16 wParam,
                      uint32 dwParam )
{
  AECHAR szBuf[] = {'H','e','l','l','o',' ',
                'W','o','r','l','d','\0'};
  AEEApplet * pMe = (AEEApplet*)pi;
  boolean handled = FALSE;

  // Decide what to do with the incoming event.
  switch (eCode)
  {
    // The application is launching.
    case EVT_APP_START:
      // Clear the display.
      IDISPLAY_ClearScreen( pMe->m_pIDisplay );
      // Display string on the screen
      IDISPLAY_DrawText( pMe->m_pIDisplay, // What
                  AEE_FONT_BOLD,           // What font
                  szBuf,                   // How many chars
                  -1, 0, 0, 0,             // Where & clip
                  IDF_ALIGN_CENTER | IDF_ALIGN_MIDDLE );
      // Redraw the display to show the drawn text
      IDISPLAY_Update (pMe->m_pIDisplay);
      handled = TRUE;

    // Application is closing
    case EVT_APP_STOP:
      handled = TRUE;

    default:
      break;
  }
  return handled;
}
```

As you can see from Listing 1-1, the code is pretty straightforward. Hello.c consists of two functions: the applet constructor `AEEClsCreateInstance` and the event handler `HelloWorld_HandleEvent`.

Invoked by the QUALCOMM BREW shell, `AEEClsCreateInstance` is the entry point of your application. It uses the helper function `AEEApplet_New` (found in AEEAppGen.c) to create an instance of your applet. This function creates an instance of the IApplet class that references your application, registering your event handler so that the QUALCOMM BREW shell can send your application system events as they occur. This function also stashes aside several important object instances in your application instance, such as a reference to the QUALCOMM BREW shell and a reference to the display.

NOTE *The AEEAppGen.c file contains a set of utility functions and the necessary wrapper to enclose your application inside an instance of the IApplet class, which is the root class for all applications. Similarly, the AEEModGen.c file contains a set of utility functions and the necessary wrapper to enclose a module inside an instance of the IModule class. Under normal circumstances, you won't need to change either of these files; however, as you learn more about QUALCOMM BREW, it's instructive to take a look at them.*

The application's event handler, `HelloWorld_HandleEvent`, needs to handle only two events: `EVT_APP_START`, which the shell sends when the application first launches, and `EVT_APP_STOP`, which the shell sends when the application ends. The event handler accepts the shell's event and arguments of the event, as well as a reference to your applet instance. In turn, the event handler handles events appropriate to the application and returns `TRUE` if the application has handled the event or `FALSE` otherwise, giving the system an opportunity to manage events that your application ignores.

Although the event handler passes a pointer to your application instance, it's of type `IApplet`, not the more specific `AEEApplet` that describes your applet. In fact, as you'll see in later chapters, you often add fields to your applet structure to carry application globals (because QUALCOMM BREW doesn't support global variables), so it's more convenient to cast the `IApplet` reference as a pointer to your application, as on line 74.

The `EVT_APP_START` event gives your application an opportunity to perform any launch-time initialization, such as creating instances of other classes, displaying the initial copyright screen, and so on. This sample application simply draws the welcome message on lines 83–92. First, it clears the display using the

IDISPLAY_ClearScreen method of IDisplay, the class that encapsulates display-related functions. It extracts a reference to the handset's display from the application instance, where the BREW function AEEApplet_New previously saved it. Next, it uses the IDisplay method IDISPLAY_DrawText to draw the message on the screen. This function lets you specify not just what text it should draw, but where it should be drawn and the bounds of the clipping rectangle that surrounds the text. It also uses two flags, IDF_ALIGN_CENTER and IDF_ALIGN_MIDDLE, to tell the display to ignore the coordinates specified and instead draw the text displayed on the center of the display. After drawing the text, it calls IDISPLAY_Update to flush the display changes to the display and mark that the application has handled the EVT_APP_START event by setting the function's return value to TRUE.

Handling application termination—which the shell indicates by sending the EVT_APP_STOP event—is much easier. Because you don't have any variables or initialized objects, you simply mark that you've handled the event and pass the result back to the system.

That's all there is to it! Of course, this is a somewhat artificial example—not only did it make minimal use of QUALCOMM BREW's capabilities, but it didn't even break up the event handler into separate functions by event, which you'll want to do in most applications. Nonetheless, this applet gives you a good idea of what goes into a simple BREW applet.

Testing Your First Application

For at least half your software development cycle, you build and test your applet using the QUALCOMM BREW Emulator, a Microsoft Windows–hosted application that lets you run your QUALCOMM BREW applets with Microsoft Visual Studio.

When you build your applet with Microsoft Visual Studio, it creates a DLL that the emulator loads when you run your application. Building your application with Microsoft Visual Studio is easy; simply select Build ➤ Set Active Configuration, choose either the Debug or Release version, and then build your application using the Build ➤ Build menu item (or pressing the F7 key).

TIP *You can use the debug and release settings to conditionally include debugging code within your code by testing the _DEBUG compiler macro. Doing this is a great way to provide debugging scaffolding within the QUALCOMM BREW Emulator while still creating relatively clean versions of your applet for demonstration purposes.*

Once you build your application, you need to configure the emulator to find both your applet's DLL and your applet's MIF. Unfortunately, this is trickier than it sounds because the emulator expects your file system to be organized in the same way that a QUALCOMM BREW–enabled handset is. This places some rather bizarre restrictions on your filenames and directory structure. For best results, you should always do the following:

- Name your applet's project directory—or the target directory where you'll be placing your built DLLs—the same as your DLL name. In this chapter's example, the hello.dll file resides in the project directory, which is named *hello*.
- Place your MIF file in the same directory as your applet DLL.
- If you're testing multiple applets, you'll need to either keep all of the project directories in one directory or copy each applet's DLL to the same directory.
- If you're testing multiple applets at once, place all of your MIFs in the same directory.

If you follow these steps, you'll find that using the emulator is far easier. To run the applet in the emulator, follow these steps:

1. Launch the application using the Build ➤ Start Debug ➤ Go menu item in Microsoft Visual Studio.
2. When Microsoft Visual Studio asks what application to run the DLL in, select the QUALCOMM BREW Emulator.
3. Within the emulator, choose the Tools ➤ Settings item.
4. On the Initial Applet Directory line, choose the directory *above* your project directory.
5. Within the emulator, choose the Tools ➤ Settings item.
6. Check the Specify MIF Directory line.
7. On the Initial MIF Directory line, select the directory that contains your application's MIF—in this example, your project directory itself.

8. Click OK.

9. Use the emulator's arrow and Select keys to choose your application.

Once you follow these steps, you'll see the screen shown in Figure 1-3.

Figure 1-3. The HelloWorld application running in the emulator

Running Your Application on Your Handset

Getting your application running on a QUALCOMM BREW–enabled handset initially seems more time consuming than useful, but it's important for you to do so as soon as you can because nothing beats running your applet on its target hardware. Testing your applets on the real hardware often and thoroughly helps you uncover implementation problems that you don't find in the emulator and helps keep you from using Microsoft Windows–specific functions in your application.

Unfortunately, getting your application to run on a handset isn't a trivial task because you need to work with the QUALCOMM staff to obtain developer access to your wireless handset. Before you can run any applications that aren't available through Mobile Shop on your wireless handset, you must do the following:

1. Sign up as a QUALCOMM BREW–authenticated developer, as described in the previous "Becoming an Authenticated QUALCOMM BREW Developer" section.

2. Be sure you've obtained a unique class ID for your applet, as discussed in the section "Writing Your First Application."

3. Buy a QUALCOMM BREW–enabled wireless handset. Although you don't need to activate the handset with a wireless carrier if your application doesn't need to access the wireless network, you should do so anyway so you can explore the other applications available through Mobile Shop. When you buy your handset, be sure to buy a serial data cable if one isn't included; you'll need one to download your application to the handset.

4. Follow the instructions on the QUALCOMM BREW extranet from the link at `http://www.qualcomm.com/brew/` and send your handset to QUALCOMM to enable developer testing support. QUALCOMM uses a proprietary in-house tool to configure your handset so that you can download your application to the handset via the data cable.

5. While you're waiting for QUALCOMM to return your handset (this typically takes a few business days), download the QUALCOMM BREW application loader from the QUALCOMM BREW extranet. This application includes support for both downloading files to your handset and running a console-style debugging log that you can use to monitor your application as it runs on the handset.

6. Use the QUALCOMM extranet to create a signature file (also called a *SIG file*) for your handset. The SIG file is a unique, cryptographically signed file that contains information from QUALCOMM and your handset's unique Electronic Serial Number (ESN).

Although all of this may sound like a needless headache—especially in comparison with competing smart phone platforms—these obstacles serve a necessary purpose. By restricting handsets so that only QUALCOMM-authenticated developers can run applications and requiring a SIG file for each handset, QUALCOMM maintains the integrity of the BREW download mechanism. Furthermore, it's nearly impossible for consumers to pirate applications, defeat network security, or tamper with the flash file systems on their handset. This insurance both increases the overall security of the wireless network (crucial to maintaining the trust of wireless carriers) and improves the consumer experience because consumers don't have easy access to tools that could render their handset inoperable if misused. Fortunately, you need to complete these steps only once for each handset you use when testing your application.

While you wait for QUALCOMM to return your handset, you can obtain one of the available tool chains to compile applications for QUALCOMM BREW–enabled handsets. Currently, you have several choices, all well documented on the QUALCOMM BREW Web site. These tool chains include a version of ARM's C and C++ compiler for the ARM chipset, which is at the core of the QUALCOMM BREW platform. If you're only dabbling with BREW, ARM offers a free, time-limited trial of its ARM compiler chain, or you can purchase a full tool chain from ARM. For more information about either, see `http://www.qualcomm.com/brew/developer/developing/armbrewpack.html`.

NOTE *At the time of this book's publication, QUALCOMM has promised to document and deliver support for using the GNU C Compiler (GCC) to compile modules for QUALCOMM BREW–enabled handsets. As you explore tool chain products for QUALCOMM BREW, be sure to check with QUALCOMM's Web site for the latest details about using GCC.*

Once you download and install your tool chain, you use a make file such as the one Microsoft Visual Studio creates when you choose AddIn Toolbar ➤ GenerateARMMakeFile. The make file uses the ARM tools to compile your sources, link them together, and convert the resulting binary file to a QUALCOMM BREW module.

TIP *If GenerateARMMakeFile doesn't work for you, close your project and open it once.*

Once you build your application for the handset, you should have three files: hello.mod, which contains your applet's object code; hello.mif, your applet's MIF file; and your SIG file. To transfer your applet to your handset, follow these steps:

1. Rename a copy of your SIG file to *hello.sig*. (You can keep the original SIG file to reuse with other applets.)

2. Create an empty folder and name it after the application, *Hello*.

3. Copy the SIG, MIF, and module files to the directory you created in the previous step.

4. Connect your handset to your computer using its data cable.

5. Launch the QUALCOMM BREW Application Loader application.

6. In response to the OEM Layer DLL Lookup dialog box, select the port to which you connected your handset and click OK.

7. Choose Module ➤ New and click Browse to select the folder you created in the second step. Enter the application name *Hello* in the lower line.

8. Click OK to transfer your applet to your phone.

That's all there is to it! After you transfer these files to your handset, you need only transfer the MOD file on subsequent builds (unless, of course, you change your MIF).

Once you transfer your applet's files to your handset, you may run the applet just as if you had downloaded it from Mobile Shop by using your phone's interface to bring up the QUALCOMM BREW application menu and select it using the direction pad.

Summary

In this chapter, you learned the following key points:

- From the developer perspective, the QUALCOMM BREW platform is a lightweight set of APIs that sits atop QUALCOMM's hardware for wireless handsets, letting you write applications in C or C++ that take advantage of the handset's features.

- Carriers are choosing the QUALCOMM BREW platform because it enables them to provide third-party applications easily and securely.

- Developers are choosing the QUALCOMM BREW platform because it lets them write applications for consumer wireless handsets.

- The QUALCOMM BREW platform includes Mobile Shop, a carrier-side application that lets consumers download your application and pay for it on their existing wireless service bill. In turn, you can monitor sales using QUALCOMM's extranet and receive regular payments for the copies of your application that consumers purchase.

- Within QUALCOMM BREW, your application is an *applet* that inherits an interface from the IApplet class. You can also write *extensions*—software libraries encapsulated as classes for applications that you and others write.

- You can begin writing your applet using a Microsoft Windows–hosted SDK consisting of Microsoft Visual Studio and the QUALCOMM SDK.

- To develop and deploy applets on wireless handsets via Mobile Shop, you must become a QUALCOMM BREW–authenticated developer via QUALCOMM's extranet at `http://www.qualcomm.com/brew/`. Becoming an authenticated developer enables you to obtain a copy of the ARM compiler necessary to build your application for the handset and enable handsets to run your application.