Chapter 1

Getting Started

Covering the essentials of a Series 60 project, plus building, deploying and running an example application

This chapter overviews the process of building, deploying and running an example Series 60 application with a Graphical User Interface (**GUI**). If you have already been involved in Series 60 development, you may want to skip this chapter. Chapter 2 builds on the overview, providing a detailed description of the project files and the development tools that are used on a regular basis.

We assume that you have installed both a Series 60 Software Development Kit (**SDK**) and a chosen Integrated Development Environment (**IDE**). Both installations are reasonably easy and are well documented elsewhere. If you do not have the necessary SDK and IDE, see the Preface for details of how to obtain them.

Topics covered are:

- **Series 60 C++ SDKs**—Different versions of the SDKs that are available and how they relate to versions of Series 60 Platform releases, and hence to specific Series 60 products.
- **Development Process Overview**—A high-level description of the process of specifying, building and running a Series 60 project for an emulator or a target device, plus a guide to the many IDE and build options available to developers.
- **Series 60 Emulators**—The features and layout of the Series 60 emulators, both debug and release versions. An overview of the similarities and differences between an emulator and a real Series 60 device.
- **Building for the Emulator**—How to build the **HelloWorld** project for the emulator using IDEs from different vendors and also from a PC command prompt.
- **Running the Emulator**—Each method of starting an emulator, and how to locate and run the example **HelloWorld** application—as both release and debug build variants.
- **Building for a Target Device**—How to build the **HelloWorld** example project for a Series 60 device as an appropriate ARM binary executable.
- **Deploying on a Target Device**—How to package up the various components of an application, options for transferring it to a Series 60 device and then how to locate and execute it.

A step-by-step overview of the development process shows you the essential steps. This is the fast-track guide to using Series 60 C++ build tools and various IDEs. You will see how to build and run an example "Hello World" application with the Series 60 emulator (version 1 and 2), plus how to build, deploy and run the application on a target device. All the information needed to build and run the **HelloWorld** example is provided within this chapter.

All the project files, source files and deployment information associated with the examples in this book are available online as noted in the Preface. If you have not already obtained the source materials, you are advised to download and install them—they will be helpful for reference as you read through this chapter.

Chapter 2 gives detailed explanations of the components that make up a typical project and the key build tools. Chapter 3 covers the fundamentals of Symbian OS and the key concepts you must understand fully to develop efficient, reliable code for Series 60 devices. In Chapter 4 the class structure of a GUI application is described and a detailed examination of typical application code is begun.

Details on debugging and testing applications are provided in Chapter 13.

Series 60 C++ Software Development Kits (SDKs)

Symbian OS is widely used in multiple smartphone platforms, such as Series 60, Series 80 and Series 90, three of the UI platforms from Nokia, and UIQ (the UI platform from UIQ Technology).

Series 60 SDKs are built upon specific versions of Symbian OS C++ SDKs released by Symbian. An SDK contains a wide range of tools, APIs, libraries and documentation to enable you to develop new applications, typically as after-market applications.

As a developer you may need to work with SDKs for more than one version of Series 60 (or even SDKs for different Symbian OS UI platforms). The most important issue is to select the correct SDK version for your chosen Series 60 product.

Nokia or Series 60 Licensees release SDKs that are suitable for development for a specific version of Series 60 Platform. That is to say, each Series 60 product is based on a particular release of the platform, and SDK releases are made to be suitable for development for a platform version. A particular SDK release may also be suitable for use with earlier versions of the platform as well. Such "backward compatibility" will depend on the version of the Symbian OS used as the basis of the release, the APIs used by the application developer and any changes that have occurred in those APIs between platform releases. Particular Licensees also may introduce additional product-specific APIs to allow developers to access the features that differentiate their Series 60 product from

those from other Licensees. For example, the Siemens SX1 smartphone includes an FM radio, so the relevant SDK may include "add-on" APIs to manipulate the radio. For maximum compatibility across different Licensee products you may want to avoid using such product-specific APIs. You may even choose to limit your use of the general APIs to those that are common and unchanged across a selected range of platform versions.

You will find the key differences between releases of Series 60 Platform described in broad detail in the Introduction to this book.

More extensive variants of Series 60 Development Platforms are available to Licensees, Competence Centers, and other software and technology partners to allow them to develop at the system level, rather than at the application level. This book, however, will focus on the publicly available SDKs.

Using Multiple SDKs

Using a single Symbian OS SDK is very easy, and the installation process will prepare it for immediate use. However, developers often need to work with SDKs for different versions of Series 60, or even SDKs for different user interface (UI) platforms. As described in Chapter 2, you can install multiple SDKs on your development PC, with a few restrictions on where they can be installed and how they are selected for use.

Development Process Overview

PC-based platform emulators are provided as part of the SDKs so that most development and testing can be performed without target hardware. Series 60 project executables can be created as debug or release variants for emulators and for target hardware (although currently some restrictions apply to on-target debugging). Additionally, when building for target devices it is possible to create executable code in various binary formats (for example, `ARMI`, `ARM4` and `Thumb`, explained later in this chapter). The compilation and linking process can be performed using command-line tools or from within a variety of IDEs. The IDEs covered here are Microsoft Visual C++ version 6, Metrowerks CodeWarrior, and Borland C++Builder 6 Mobile Edition and Borland C++BuilderX Mobile.

Symbian devised a method of specifying development projects in a platform-neutral way. Two universal project files can be created (`projectname.mmp` and `bld.inf`), where `projectname` is the name of the component or application to be developed (**HelloWorld** for our example project). These two text files can then be used as a starting point for any of the build options, IDEs and platform variants.

The `bld.inf` file specifies the names of all the project component(s) to be built, with each component specified in its own `.mmp` file. Both types of file are plain

text, and often you will simply have a single .mmp file that defines the application you are creating. If the project consists of multiple components, such as the application itself and specific function libraries, then each component would have its own .mmp file. Each of the libraries plus the application would have an .mmp file, and each filename would be listed in the bld.inf file for the project. The syntax of .mmp and bld.inf files is detailed in Chapter 2.

In the **HelloWorld** example, there are only two project specification files: bld.inf and helloworld.mmp. Using these two files, any platform-specific project and command files required can be created.

Typically you employ a Symbian tool called bldmake, using the two project specification files as input, to generate a command file called abld.bat. You can then use abld.bat, from the command prompt, to perform a number of project-related actions. For example, abld.bat can be used to generate platform- and IDE-specific project makefiles. If the project source code exists, and is complete, abld can be used to build the project for one or more platforms.

Since most development projects are built and run from within an IDE, you would usually create the project files suitable for your chosen IDE.

In the case of Microsoft Visual C++, you use abld at the command line to create the HelloWorld.dsp and HelloWorld.dsw project files. The .dsw file is the workspace file to be opened from within the IDE, and it may reference one or more .dsp files.

For Metrowerks CodeWarrior you can either create the project file from within the IDE in the usual way or import the .mmp file directly into the IDE. The import process will create the CodeWarrior specific project (HelloWorld.mcp) file required.

Similarly, the Borland C++ Builder Mobile Edition IDEs can perform an equivalent import task to the Metrowerks IDE, but by importing the bld.inf file instead of the .mmp file. Borland C++BuilderX, for example, will create a project file called HelloWorld.cbx.

Opening the IDE-specific project file will then allow you to develop, build, run and debug the application with full IDE support.

For developers who prefer working at the command-line level, abld can also be used to compile and link from a command prompt.

All of the methods for creating the IDE (or command-line) specific project files outlined are described in detail later in this chapter. In addition, Figure 1–1 illustrates the use of the two generic Symbian OS project files to generate the required platform-specific project files, either via IDE import options or using the Symbian tools.

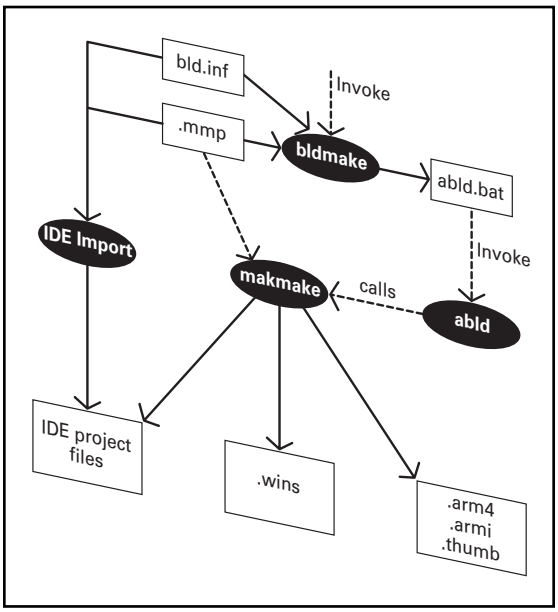

Figure 1–1 Generation of IDE and platform-specific project files from generic Symbian files.

Using an IDE versus Command-Line Tools

The different options currently available to you for working with Series 60 development projects are summarized in Table 1–1. More options are becoming available all the time from Symbian, Nokia and the development tool vendors Borland and Metrowerks. However, it is likely that the Microsoft development tools, though currently viable and still widely used, will no longer be supported in the not-too-distant future.

There are currently considerable differences between the capabilities of the various IDEs from Microsoft, Borland and Metrowerks. Development to enhance the latter two development environments is ongoing. EMCC Software Ltd uses IDEs from all three vendors as the basis of its development activities. We also use the command-line tools every day—not because we want to but because we currently have to. This is primarily for building for target devices and for automating overnight builds. The C++BuilderX and CodeWarrior IDEs can now build for target devices—but not all essential build operations are covered by some variants of the IDEs.

At the time of writing, the easiest and most generic starting point for any new Series 60 project is to define a pair of `bld.inf` and `.mmp` project files. At EMCC Software, these files are used for generation of any IDE-specific project files and for use during any command-line builds needed. This approach is taken throughout the book. It is also the approach currently adopted by every single example project provided by both Symbian and Nokia in their SDKs and in the

documentation they supply. Using an IDE exclusively is not currently possible on a day-to-day basis. Things will change before very much longer, but these are the current facts of life for Symbian OS and Series 60 developers.

Table 1–1 PC-Based Development Options Summarized

Option	Description
Command line building	Using the Symbian OS tools combined with compilation and linking using the Microsoft Visual C++ compiler/linker for emulator builds invoked from a command line. Emulator can also be invoked from the command line. Source-level debugging on the PC emulator requires working from within an IDE. The GNU C++ cross compiler and linker used for ARM target device builds invoked from the command line.
Microsoft Visual C++ IDE	Compilation and linking using the Microsoft Visual C++ compiler/linker for emulator builds from within the IDE. The GNU C++ cross compiler and linker used for ARM target device builds invoked from a command prompt.
Borland C++ IDE	Two IDE options are currently available; both use the Borland C++ compiler and linker for emulator builds from within an IDE. The GNU C++ cross compiler and linker used for ARM target device builds invoked from the command line.
Metrowerks CodeWarrior C++ IDE	Using the Metrowerks compiler and linker for emulator builds from within the IDE. The GNU cross compiler and linker for ARM target device builds, either invoked from the command line or directly from within the CodeWarrior IDE.

Through the rest of this chapter, the **HelloWorld** example project is used to illustrate all the steps involved in the development, debugging, and deployment process for the Series 60 emulator.

All of the other files required for the example application are provided—for example, the header (`.h`), source (`.cpp`) and user interface resource (`.rss`) files.

You can test the application using the Series 60 emulator, either started from within an IDE or run from a command prompt. However, debugging an application on the emulator must be performed from within an IDE.

After developing, running, testing or debugging an application on the emulator you typically will want to build and run it on a target device. So you will then be shown how to build the example application for a target device, how to deploy it and then run it on target Series 60 hardware.

Series 60 Emulators

Development, debugging and initial testing of Series 60 applications is usually carried out on a PC-hosted emulator that provides a Microsoft Windows-based implementation of a Series 60 device. In most cases you will discover that the emulator-based development process closely mimics the operation of an application running on a real device; so that the majority of your development work can take place even before hardware is available. The exact appearance of the emulator may vary from the figures provided and will depend on the target platform you are working with, the version of Series 60 you are using, and the selected IDE. For example, the fascia bitmap may have been changed to closely resemble a particular Series 60 device from a Licensee. In addition, buttons and other interaction elements may be moved or added to emulate the configuration of a manufacturer's actual device. Also, the applications available on the emulator will depend on the platform version and the device manufacturer's preferences.

Some differences between an emulator and a real device cannot be overcome. Real Series 60 devices will have hardware accessories such as cameras and other features such as vibration feedback. Thus, at some point, hardware will be necessary for development and testing. In addition, PC-based emulators do not accurately mimic issues of precise timing, application performance and memory management.

For PC-based development, the edit/compile/build cycle is based on a Microsoft Windows-hosted development toolset. However, instead of linking and building against Win32 or MFC libraries, developers link and build against the headers and PC-format libraries installed by the Series 60 SDK. The resulting Windows-format binary executable is then run under the PC-hosted emulator.

During development, the project file for the specific IDE manages all linking and building details. It also ensures that all outputs from the build, and other required resources, such as application resource files, are built into the appropriate location for running or debugging under the emulator.

Referring to Figure 1–2, the Series 60 display is logically divided into three areas: status pane, main pane and control pane. (See the "Series 60 UI Style Guide" provided with the Series 60 SDK documentation for a comprehensive description of user interface elements and standards.)

The status pane is the graduated (blue, on the emulator) bar near the top of the screen plus the area above it. The main pane is the middle section of the screen between the status pane and the soft key labels at the bottom of the screen. The control pane is the area immediately below the main pane and includes the soft key labels.

The status pane may display information for the current application, as well as general information about the device status, such as the signal strength and battery charge. It is visible in most situations, but sometimes it may be hidden. Many games, for example, will use the whole screen.

Figure 1–2 Series 60 emulator.

The main pane is the principal area of the screen where an application can display its data. Typically, this area, also referred to as the client rectangle, is fully occupied by an application's data display.

The control pane occupies the bottom part of the screen and displays the labels associated with the two soft keys and a scroll indicator when required. Like the status pane, the control pane can also be hidden at times. In such situations it is within Series 60 style guidelines to have the user assume the availability of the **Options** menu (the default label for the left soft key), even though it may not be visible. (See the "Nokia Series 60 Games UI Style Guide" provided with the Series 60 SDK documentation.)

The two buttons below the control pane are the left and right soft keys and are used to select the currently associated **Options** menu or labeled action. The four-way navigation key will scroll up, down, left, right, or will select if pressed (clicked) in the center.

You can interact with the emulator via the PC mouse or cursor keys for navigation around the objects on the display. It is possible to use mouse clicks to select folders and other displayed objects directly rather than via clicking on

the four-way navigation key (on the emulator only). For data entry the PC keyboard can be used, or you may click on the twelve-way keypad on the emulator fascia. Therefore the interaction with the emulator is close to, but not exactly the same as, using a real Series 60 device—there is no pen input on a real device, so all movement and selection is through cursor (joystick) navigation.

Building for the Emulator

Since Series 60 applications can be built from a command prompt or from within an IDE, we have detailed both methods here. We'll start by building the project to run under the PC-hosted emulator (that is, for an x86 instruction set) using the C++ compiler supplied with the IDE. We almost always use a debug build, so that symbolic debug information and memory-leak checking are available (checking for memory dynamically allocated on the heap that is not released correctly).

Building from the Command Line

Open a command prompt and change to the drive/folder that contains your Series 60 SDK. Navigate to the folder where the project definition (`helloworld.mmp`) and component description (`bld.inf`) files are located—for example:

```
\Symbian\Series602_0\EMCCSoft\HelloWorld\group
```

for a Series 60 2.x project

or

```
\Symbian\6.1\Series60\EMCCSoft\HelloWorld\group
```

for a Series 60 1.x project

and type:

```
bldmake bldfiles
```

After a second or two this command completes without any visual output. It uses the `bld.inf` and `helloworld.mmp` files to generate a new file: `abld.bat`. This command file is always generated in place, as required. Unlike the `bld.inf` and `.mmp` files, `abld.bat` is not portable between different IDEs and should never be edited by hand.

To compile and link the project, type:

```
abld build wins udeb          —for Visual C++
abld build winsb udeb         —for Borland C++
abld build winscw udeb        —for CodeWarrior
```

The `abld` command will build the project (in other words, compile and link) for the Series 60 emulator (the `wins`, `winscw` or `winsb` variant) with debugging (`udeb`—Unicode debug) information included in the binary executable.

Building from an IDE

Projects, such as our example **HelloWorld** application, normally are built and run from within an IDE, so we need to create IDE-specific project files from the `bld.inf` and `HelloWorld.mmp` files. For Visual C++ this must be performed from the command line, using tools supplied by Symbian. For Borland and CodeWarrior this is optional, since both IDEs can import either the `bld.inf` or `.mmp` file, respectively, to create the IDE project files.

When working from the command prompt it may be necessary to create the `abld.bat` file if it does not exist already, or recreate it if the `.mmp` file or `bld.inf` file has changed. At a command prompt you create the `abld` command file by typing:

```
bldmake bldfiles
```

Building Using Microsoft Visual C++ IDE

Open a command prompt and navigate to the drive/project folder for the **HelloWorld** project and then type:

```
abld makefile vc6
```

This will create project and workspace files (`helloworld.dsp` and `.dsw` files) suitable for Microsoft Visual C++. They will be located under the `\Epoc32\Build` sub-folder structure; the complete path will depend on the location of your SDK, for example:

```
\Epoc32\Build\EMCCSoft\HelloWorld\HelloWorld\Wins
```

By opening the workspace file (`helloworld.dsw`) in Visual C++, you can compile or link the application, either by pressing **F7** or via the IDE menu option, **Build|Build HelloWorld.app**.

Building Using Borland C++IDE Builder 6

If you are using Borland C++Builder 6 Mobile Edition, which is based on C++Builder 6 Personal Edition with the Mobile plug-in, you can simply import the `bld.inf` file for the **HelloWorld** project into the IDE. Use the **File|New|Other** menu option. Then select the **Mobile** tab in the resulting dialog: **Import Mobile Application**. Browse to the location of the component description (`bld.inf`) file and open it.

Use **Ctrl+F9**, or **Project|Make** from the menu, to build. To build and run, use **F9** or **Run|Run** from the menu. Note that **F9** or **Run|Run** will cause a project rebuild

each time! To just run the emulator, use **Tools|Mobile Build Tools|Run Emulator**. You may be prompted to save a number of project-related files, for example, Borland project (`.bpr`), Borland project group (`.bpg`) files. These files will be saved in the same folder as your `bld.inf` file.

It is also possible to execute individual **abld** commands and run other SDK tools such as **aifbuilder** and **sisar** from the **Tools|Mobile Build Tools** menu. These tools are described in Chapter 2.

Building Using Borland C++BuilderX

When using any of the commercial C++BuilderX products, you can simply import the `bld.inf` file for the **HelloWorld** project into the IDE.

Use **File|New** and click the **Mobile C++** tab in the Object Gallery. Select **Import Symbian C++ Project**. Select the correct Series 60 SDK from the drop-down list, browse to the location of the project `bld.inf` file. Press the **Next** tab, give the project a name, press the **Finish** tab and the project will open. Press **Ctrl+F9** or select **Project|Make Project**. To run, Press **F9**, or select **Run|Run Project** or use the toolbar tab to **Make and Run** the project.

Building Using CodeWarrior IDE

If you are using Metrowerks CodeWarrior for Symbian (Personal v2.5, other editions may vary slightly), you can simply import the `HelloWorld.mmp` file using the IDE menu option, **File|Import Project from .mmp File**.

This runs a project conversion wizard. Select the SDK to use with this project, select (or browse to) the `.mmp` file, and select a platform of `WINSCW` (or all by leaving it blank). The build variant will default to `UDEB`. Use **F7** or **Project|Make** from the menu to build the project.

The CodeWarrior Project files (`.mcp, .xml, .resources` and `.pref` files) are created automatically in the same directory as the `HelloWorld.mmp` file.

Alternatively you can create a CodeWarrior IDE project from the command line. To do this, run `bldmake bldfiles` as described; then, to generate a CodeWarrior IDE project, use:

```
abld makefile cw_ide
```

This creates an importable project file `HelloWorld.xml` in the directory:

```
\Epoc32\Build\EMCCSoft\HelloWorld\HelloWorld\Winscw
```

You can now use CodeWarrior to import this file and to generate a native project (`.mcp`) file. Choose the **File|Import Project** menu option, select the `HelloWorld.xml` file, and choose a name for the project (such as **HelloWorld** again). CodeWarrior will now generate and load the project, which you can build, run, debug and so on, using the normal IDE commands.

Running the Emulator

In a Series 60 SDK two versions of the emulator executable are available: a version built containing symbolic debugging information, and another built as a release variant. The release emulator is limited to evaluation and demonstration of applications—it starts up considerably quicker because of the absence of the debugging information.

Both versions are called `epoc.exe`, but they are located in their own subdirectories. The name `epoc` is historical—it was the name of the operating system prior to Symbian OS.

In normal development activities, it is usual to use the debug variant of the emulator. Depending on your choice of IDE, you may be able to run the debug version normally or in "debug mode." To be able to run the same "debug emulator" in two modes may seem a little confusing at first.

Sometimes you may want to start the emulator, locate the application and run it (as described later) simply for testing purposes. If a serious error occurs, the emulator and application will shut down in a controlled way.

Other times you may want to put a breakpoint in your code at a specific point where you think a problem exists, and then have the IDE run the emulator in "debug mode." You then locate the application and run it as before. Suitable interaction with your application will cause the breakpoint in the code to be reached. At that point, the source code will be displayed in the IDE along with all of the symbolic debug information associated with the application. Then you can use the debugging features of the IDE to step through sections of code, in a controlled manner. All the while you are able to view the application source code, data, call stack and other debug-related information displayed by the IDE to assist you in tracking down errors in code or logic.

The appearance of a typical Series 60 emulator is shown in Figure 1–3. Debugging an application under an emulator using the Microsoft Visual C++ IDE is illustrated in Figure 1–4.

Emulator Executable Locations

For a Series 60 1.2 SDK the release build emulator is typically located under the following:

For Visual C++

```
\Symbian\6.1\Series60\Epoc32\Release\wins\urel\epoc.exe
```

For Borland C++

```
\Symbian\6.1\Series60\Epoc32\Release\winsb\urel\epoc.exe
```

For CodeWarrior

```
\Symbian\6.1\Series60\Epoc32\Release\winscw\urel\epoc.exe
```

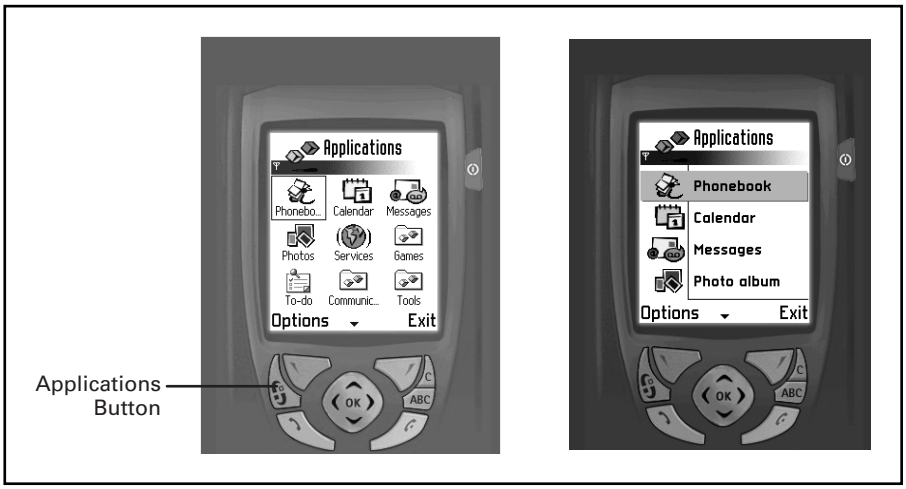

Figure 1–3 Applications grid and list views of Series 60 Platform 1.2 debug emulator.

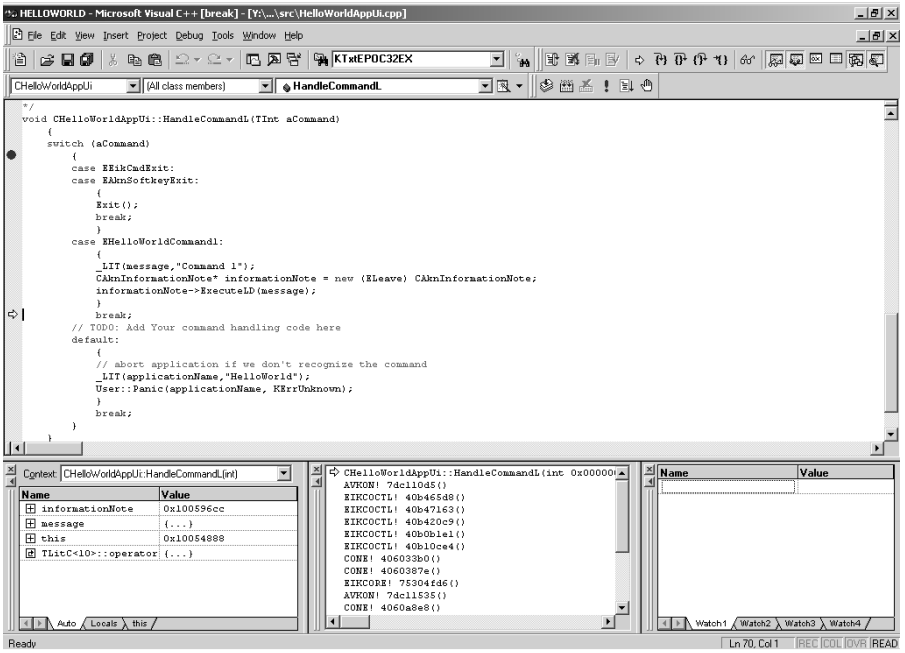

Figure 1–4 The HelloWorld application on the emulator in debug mode under the Microsoft Visual C++ IDE.

The debug build emulator is typically located under:

For Visual C++

```
\Symbian\6.1\Series60\Epoc32\Release\wins\udeb\epoc.exe
```

For Borland C++

```
\Symbian\6.1\Series60\Epoc32\Release\winsb\udeb\epoc.exe
```

For CodeWarrior

```
\Symbian\6.1\Series60\Epoc32\Release\winscw\udeb\epoc.exe
```

The exact paths will depend on the options you choose during the installation of the SDK. In the case of the Series 60 2.x SDK the paths to the emulator will be very similar—for example: `\Symbian\Series602_0\Epoc32\release\wins\udeb\epoc.exe`.

Figure 1–3 shows the Series 60 1.2 emulator. It starts up showing the Applications main menu as either a grid view or a list view.

The Series 60 2.0 emulator starts with a mock-up of the phone application, and you have to navigate to the applications menu by pressing the applications button shown in Figure 1–3. Whatever version of Series 60 Platform you are using, always specify the debug (`udeb`) version of the emulator executable as the default for development projects—for example: `\Epoc32\Release\wins\udeb\epoc.exe`.

Emulator Debug Mode

When running the application in debug mode under the emulator, the source code, function call stack, variable information and so on are shown as soon as the breakpoint in the code is reached, as shown in Figure 1–4. The emulator window itself may disappear (it is minimized) if the application code is not at a point where user input is required.

Running the Emulator from a Command Prompt

To run the debug emulator from the command line, open a command prompt, change to the folder in your Series 60 SDK where the `epoc.exe` application is located (for example, `\Symbian\6.1\Series60\Epoc32\Release\wins\udeb`) and type the following:

```
epoc
```

This will start the debug emulator and you can then locate and run your application, but not in debug mode. To debug an application you need to run the emulator in debugging mode and this can only be done from within an IDE. To run the release emulator enter the following:

```
epoc –rel
```

Running the Emulator from the Visual C++ IDE

From within the Visual C++ IDE you can start the debug version of the emulator by pressing **Ctrl+F5**, or from the menu use **Build|Execute Epoc.exe**. This will run the emulator in non-debug mode. Alternatively, you can use **F5** or select **Build|Start Debug|Go** from the menu to run the emulator in debug mode.

The first time you run the emulator for a Visual C++ project, a dialog will appear asking you to supply the name of the executable. Navigate to `epoc.exe` in the folder `\Epoc32\Release\wins\udeb` in the root of your SDK.

Running the Emulator from the Borland C++Builder 6 and C++BuilderX IDEs

You can start the debug version of the emulator using the **Tools|Mobile Build Tools** menu option, then select **Run Emulator**. Alternatively you can use the **Run|Run** menu option (**F9**), but it will cause a project rebuild each time—this can be a lengthy and time-consuming process! Using this option, you will need to cancel the build dialog ("Compiling") before the emulator will start up.

If you are running the emulator from C++BuilderX, use the **Run|Debug Project** menu option or press **Shift+F9**. If you wish to rebuild the project and start the emulator, select **Run|Run Project** or press **F9**.

Running the Emulator from the CodeWarrior IDE

Select the **Project|Run** menu option or press **Ctrl+F5** to run the emulator. Press **F5** or use the **Project|Debug** menu option to run the emulator in debug mode.

Locating and Running the Application

Navigate to, and select the **HelloWorld** application by clicking on the image of the cursor keys on the emulator fascia bitmap, or by using the PC keyboard cursor (arrow) keys. Click on the **Selection** button (in the middle of the cursor controls) to start the application.

Applications that do not have their own specific icon (as specified in an `.aif` file, detailed in Chapter 2) will be given a default icon, which looks like a piece of a jigsaw puzzle, by the system.

Under some SDK/IDEs (for example, versions of Borland and CodeWarrior), the application you have built may be located in a folder called "**Other**" rather than on the main desktop. If so, navigate to and select the **Other** folder and then open it by clicking on the Selection button. Navigate to and select the **HelloWorld** application and click on the **Selection** button to invoke the application.

The **HelloWorld** application will run and should appear as shown in Figure 1–5.

Figure 1–5 The "Hello World" application.

Debugging the Application

Alternatively, you could run the application on the emulator from within an IDE in debugging mode—the procedure will vary depending on the IDE in use. Typically you would first set a breakpoint at an appropriate point in the source code.

Start the emulator in debugging mode. Since it is the application (essentially a dynamic link library—**DLL**) that will be debugged, not the emulator itself, navigate to and run the application as described earlier in "Locating and Running the Application." The application will start up, and then execution will stop at the breakpoint you set earlier. You can the use the facilities of your chosen IDE to step through the execution of the application source code.

Further IDE Help

Further explanation of the various IDE functions is beyond the scope of this chapter, so for more details refer to the IDE help information, available through the **Help** menu option.

For Microsoft Visual C++ this is accessed through the **Help|Contents** menu option, provided you installed MSDN with your IDE.

For Borland C++Builder 6 this is accessed through the **Help** menu option. You will find a separate **Help|Borland C++ Mobile Edition Help** page as well as the standard Borland help files. For C++BuilderX, select **Help|Help Topics** and choose **Mobile Development**.

For Metrowerks CodeWarrior there is a **Help|Online Manuals** menu option that contains lots of valuable information on working with Symbian OS.

Additional specific IDE information can be obtained online—for example, for Visual C++ information go to **http://msdn.microsoft.com/**, for Borland C++ go to **http://bdn.borland.com/** and for CodeWarrior go to **http://www.metro-werks.com/MW/Develop/Wireless/**.

Building for a Target Series 60 Device

Building for a Series 60 device with the Visual C++ IDE must be performed at the command prompt using the `abld` command, as described next. However, CodeWarrior and C++Builder allow building for a target device from within the IDE.

Building for a Series 60 device based on an ARM processor on a PC requires the use of a suitable PC-hosted cross compiler (for example, the GNU **gcc** C++ Compiler, as supplied with the Series 60 SDK) to build the executables in a suitable ARM binary format.

When building the project for a Series 60 device you would typically use a release build, since that is what you would do to create a final, deliverable application.

To build for target hardware, open a command prompt window and navigate to the group directory for the GUI **HelloWorld** project, then enter the following commands:

```
bldmake bldfiles
abld build armi urel      —For Visual C++ and CodeWarrior
```

or

```
abld build armib urel     —For Borland C++
```

This will cause `abld` to invoke the build (cross-compilation and linking) system to produce an `armi` (ARM Interworking) release (`urel`—Unicode release) build of the application for execution on a target device using the **gcc** tool chain.

There are currently three build variants for ARM based devices—`ARMI`, `ARM4` and `Thumb`. `ARMI` executables will work with the other two build variants. Typically you should build the `ARMI` binary executable variant for compatibility with the maximum number of real devices. `ARM4` builds give maximum performance at the expense of increased code size. `Thumb` builds will reduce the code size at the expense of a slight reduction in execution speed.

The build steps actually include C++ compilation, linking, resource compilation, and production of the application information (`.aif`) file—the file that contains the application icon and other specific details.

When building for a target device a Symbian OS-specific tool called **petran** is automatically invoked behind the scenes. **Petran** translates `HelloWorld.app` into a form suitable for loading at runtime by the Symbian OS executable loader.

If you are using the CodeWarrior IDE to build for a Series 60 device, use **Project|Set Default Target** either to select the required target (for example, `ARMI UREL`) or to choose **Build All**, and then build with **Project|Make**.

With C++Builder X, you need to select the **Project|Properties** menu item and then select the **Symbian settings** tab. From this dialog you can change the **Platform** and **Build** options to `ARMI` and `UREL`, respectively. *Note:* At the time of writing `ARMI` was the only target Platform option available.

Pressing **Ctrl+F9** will make the project, creating the installation package (`.sis`) as part of the project; installation packages are described in the next section.

The executable and data files (`HelloWorld.app`, `HelloWorld.rsc` and `HelloWorld.aif`) will now be located in a folder such as `\Epoc32\Release\Armi\Urel`.

For testing on a Series 60 device, all of these files have to be transferred to a device and located in a folder called `\System\Apps\HelloWorld\`.

File transfer to a device can be performed by copying to a memory card, or via a USB cable, if the Series 60 device manufacturer has included appropriate support in their product. Typically though, the file transfer is performed by packaging all the application files into a special installation file. The next section describes how this is achieved.

Deploying on a Target Device

Applications are delivered to target hardware in the form of a **Symbian Installation System** (`.sis`) file. A `.sis` file is a single compressed archive file, containing all of the files required for installation, plus optional information about the installation process. The Symbian Installation System provides a simple and consistent user interface for installing applications, data or configuration information onto devices based on Symbian OS. Developers (or end users) install components, packaged in `.sis` files.

Production of `.sis` files can be performed using the interactive **`sisar`** tool provided with the Series 60 SDK. **`Sisar`** packages all the application files into one `.sis` file for ease of installation onto target hardware. Alternative methods for producing installation files are described in Chapter 2.

Everything required to make an installation (`.sis`) file is provided with the example **HelloWorld** project—under the `\install` folder. In this example project we will use a special installation package source file called `HelloWorld.pkg` and a tool from Symbian called `makesis.exe`.

Building a SIS Install File

After building the `armi` release version of the **HelloWorld** application, as described above, you need to package up the application components into an installation package (`.sis`) file. Open a command prompt and navigate to the SDK folder for the **HelloWorld** project. Change to the `\install` folder, then build the `.sis` file by typing:

```
makesis helloworld.pkg
```

A successful build will produce an output message such as "`Created helloworld.sis`". The installation package (`.sis`) file will have been created in the `\install` folder. Now you need to transfer it to the device, as described in the next section.

SIS File Installation

You may choose among three potential installation options, depending on the device you are using, and other facilities available to you—for example, whether you have access to a PC with infrared or Bluetooth capability, or access to appropriate software based on Symbian Connect (Nokia PC Suite, for example, or a branded equivalent provided by the device manufacturer):

- Installation through the invocation of a `.sis` file located on a PC, with subsequent application installation on to the Series 60 device through an infrared or Bluetooth session between the PC and the target device, established via software such as Symbian Connect.
- Installation by transfer of a `.sis` file through **OBEX** (OBject EXchange), over infrared or Bluetooth, from another device such as a PC, Symbian OS phone or any OBEX-enabled device. This process will be managed via the **Messaging** application, which intercepts the file attached to the message—when you open up the message, it will automatically start the application installation process on the phone.
- Alternatively, `.sis` files can be sent as email attachments. Application installation is again managed via the **Messaging** application on the phone. When you open the message, it will automatically start the installer.

The first two options depend on establishing a connection between your development PC and the Series 60 device. The device manufacturer typically supplies suitable communications software, and you will need to refer to the specific instructions supplied with the connection software.

After installation a much-reduced version of the `.sis` file remains on the Series 60 device to control the uninstallation of the application, if required, using the application "**Manager**." This reduced `.sis` file contains only the information required to uninstall the application and is typically very much smaller than the original file.

Often the original `.sis` file may still exist on the device, if it was delivered as a message attachment and the original message has not been deleted from the **Messaging** application's **Inbox** folder.

Running on a Target Device

Transfer the `helloworld.sis` file provided to the target hardware, using one of the methods described above. After the transfer you will be offered the chance to install the application on the device. To run the application follow the procedure outlined in "Locating and Running the Application" earlier in this chapter. You will be reassured to find that locating and running the application on a target device is identical to the process on the emulator—with one small difference: the application will not be located in an "**Other**" folder.

Summary

This chapter has described the whole process of building, deploying and running a simple GUI application on the Series 60 emulator and on a real target device. The skills you have gained here will be used on a daily basis as you continue to develop for Series 60.

In the next chapter, you will build on this knowledge and look in detail at the creation of a simple Series 60 application. You will learn about the composition of project files, examine application information files and understand how to produce installation package files. Some key additional SDK tools that are used on a regular basis will also be described.