
Introduction to Series 60 Applications for C++ Developers

Version 1.0
August 2002

Table of Contents

1.	Introduction.....	6
1.1	Purpose and Scope.....	6
2.	Series 60 Platform	6
3.	Symbian OS 6.1	6
4.	Series 60 C++ SDK.....	6
5.	Example C++ Applications	7
5.1	The Hello World Console Application.....	7
5.1.1	Build and Run from the Command Line	7
5.1.2	Build and Run from the IDE.....	8
5.1.3	Console Test Harnesses	8
5.2	The Hello World GUI Application	9
5.2.1	Build and Run from the Command Line	9
5.2.2	Build and Run from the IDE.....	10
5.3	Application UI Elements	11
5.4	The Symbian OS Application Framework	12
5.5	Series 60 Application Framework	12
5.6	Alternative GUI Designs	13
5.7	UI Style	14
5.8	Series 60 Application Wizard	14
5.9	GUI Application Project Files	14
5.9.1	bld.inf	14
5.9.2	HelloWorld.mmp	15
5.10	Resource Files	16
5.11	Source Files	16
5.12	Header Files	17
6.	Executable Files and Locations.....	18
6.1	HelloWorld Console Application Files	18
6.2	HelloWorld GUI Application Files	18
6.2.1	AIF Files.....	19
6.2.1.1	Icons	19
6.2.1.2	Captions	19
6.2.1.3	MIME support.....	20
7.	Building for a Series 60 Device.....	20

8.	Implementing On a Series 60 Device.....	20
8.1	SIS File Installation.....	20
8.2	SIS file creation	21
9.	Developer Resources	22
9.1	Forum Nokia.....	22
9.2	Symbian Developer Network.....	22
9.3	Books	23

Legal Notice

Copyright © Nokia Corporation 2002. All rights reserved.

Reproduction, transfer, distribution or storage of part or all of the contents in this document in any form without the prior written permission of Nokia is prohibited.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Other product and company names mentioned herein may be trademarks or tradenames of their respective owners.

Nokia operates a policy of continuous development. Nokia reserves the right to make changes and improvements to any of the products described in this document without prior notice.

Under no circumstances shall Nokia be responsible for any loss of data or income or any special, incidental, consequential or indirect damages howsoever caused.

The contents of this document are provided "as is". Except as required by applicable law, no warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, are made in relation to the accuracy, reliability or contents of this document. Nokia reserves the right to revise this document or withdraw it at any time without prior notice.

Glossary Definitions

The following terms and abbreviations are used within this document.

Term	Meaning
API	Application Programming Interface
AVKON	Series 60 extensions and modifications to UIKON and other parts of the Symbian OS application framework
IDE	Integrated Development Environment
GUI	Graphical User Interface
OBEX	Object Exchange – transfer of objects such as files or data between two devices e.g., by Infrared or Bluetooth
SDK	Software Development Kit
UI	User Interface
UIKON	A UI and control framework common to all Symbian OS devices

1. Introduction

1.1 Purpose and Scope

This document is an introduction to creating two different types of “Hello World” applications in C++ for Series 60 Platform 1.0. Series 60 devices are based on Symbian OS 6.1, an open, robust, multitasking operating system designed for data-enabled mobile phones. Symbian OS is written largely in C++, which is also a strong development choice for third parties.

An introduction will be provided to Series 60 Platform 1.0, Symbian OS 6.1. The Series 60 SDK and associated developer tools along with information about further reading materials. It is assumed that the reader has no prior specific knowledge of Symbian OS, but is familiar with C++ and has obtained the Series 60 Software Development Kit (SDK) for C++.

2. Series 60 Platform

Series 60 Platform is a complete smartphone reference design, including a host of wireless applications. The platform builds on the Symbian operating system (Symbian OS), complementing it with a configurable graphical user interface library and a comprehensive suite of reference applications. A set of robust components and many varied APIs are provided in Series 60 Platform. The APIs supplied are used extensively by the suite of “standard” applications but they are designed to be re-used by third-party application developers as well.

3. Symbian OS 6.1

Central to the success of Series 60 Platform is Symbian OS; it is the foundation of the product.

Symbian OS is a 32-bit multitasking operating system, wherein events often happen asynchronously and applications are designed to interact with one another. For example, a phone call may interrupt a user composing an email message, a user may switch from email to a calendar application in the middle of a telephone conversation, or an incoming SMS may trigger the user to access the contact database and forward the SMS on. By complying with the platform architecture and software design guidelines, application designers can routinely manage such occurrences in the daily lives of smartphone users.

4. Series 60 C++ SDK

Series 60 Platform ships with its own SDK based on the Symbian SDK. The APIs enable third-parties to develop new Series 60 applications written in C++ for inclusion in their products or as value-added, after-market applications.

The Series 60 SDK provides documentation, tools, and sample code to assist developers, along with a Microsoft Windows-hosted emulator. The SDK is essential for developing, testing, and debugging C++ applications.

5. Example C++ Applications

An explanation follows of two “Hello World” applications for Series 60 devices, a console application, and a basic GUI application. To build the applications, Microsoft Visual C++ 6.0 (Service Pack 3) is required and, since the SDK build system uses Perl scripts, it is necessary to have Perl installed.

5.1 The Hello World Console Application

This first “Hello World” example is a console application and consists of a single executable file with a file name extension of .exe. In Symbian OS, such executables are used for two main purposes; as servers with no user interface or as test harnesses with very simple text-based interfaces. Typical sophisticated GUI applications make use of the application framework and user interface libraries. A GUI-based “Hello World” application is described later in this document.

5.1.1 Build and Run from the Command Line

Open a command prompt and change to the drive that contains The Series 60 SDK. Navigate to the folder that contains the project code, e.g.,
`\Symbian\6.1\Series60\Epoc32Ex\Basics\HelloWorld`

This folder contains three files:

HelloWorld.cpp - the source file
HelloWorld.mmp - the project definition file
Bld.inf - the component definition file

To build the example, type:

bldmake bldfiles

This will generate a new file, i.e. ABLD.BAT; this command file is always generated as required and should never be edited.

To compile and link the project enter:

abld build wins udeb

This will build the project for the Series 60 debug emulator. To run the application:

At the command prompt, navigate to the folder that contains the Helloworld.exe application, e.g.,

`\Symbian\6.1\Series60\Epoc32\Release\wins\udeb`

At the command prompt, type:

helloworld

The Series 60 emulator starts and the application will appear as shown in Figure 1:



Figure 1: The console emulator

Note that the application simply waits for any key press on the PC. Pressing any key will complete the application and close the emulator.

5.1.2 Build and Run from the IDE

Normally projects such as the HelloWorld application are built and run from within the Microsoft Visual C++ 6.0 IDE as follows.

If the ABLD.BAT file does not already exist (or if the .mmp file or bld.inf file have changed) the build command file must be generated by typing:

```
bldmake bldfiles
```

followed by:

```
abld makefile vc6
```

This will build the project and workspace files (.dsp and .dsw) for Visual C++. They will be located under the `\epoc32\build` sub-folder structure e.g.,

```
\epoc32\build\symbian\6.1\series60\epoc32ex\basics\helloworld\helloworld\wins
```

By opening the workspace file `HelloWorld.dsw` in Visual C++, the application can be built within the IDE by pressing F7 and then run using Ctrl+F5. The console emulator will start up automatically with the HelloWorld application running.

5.1.3 Console Test Harnesses

Note that the project code in the `HelloWorld.cpp` file is very simple and implements a single function called `doExampleL()`, called from `callExampleL()`, a function defined in a header file `CommonFramework.h`. This file is provided as part

of The Series 60 SDK and is ideal as the basis for console-based test harnesses used during general development work. Console applications and the emulator start up very quickly because no GUI libraries have to be loaded. The key disadvantage of console applications (apart from the simple interface) is that only one application can be running in the emulator at once.

5.2 The Hello World GUI Application

To aid maintainability and flexibility, Symbian OS applications are typically split into two main parts, the engine, also known as the application model, and the user interface (UI). The engine encompasses the data structures needed to represent the applications' data, the algorithms, and any data persistence. The UI is typically sub-divided into on screen representation(s) of the data and a handler that determines the overall behavior of the application. The four key classes in a Symbian OS GUI application are discussed in more detail later in this document.

5.2.1 Build and Run from the Command Line

Open a command prompt and change to the drive that contains The Series 60 SDK. Navigate to the folder that contains the project definition (.mmp) and component definition (bld.inf) files e.g., `\Symbian\6.1\Series60Ex\HelloWorld\group`.

To build the example, type:

```
bldmake bldfiles
```

This will generate a new file, i.e. **ABLD.BAT**; this command file is always generated as required and should never be edited.

To compile and link the project, type:

```
abld build wins udeb
```

This will build the project for the Series 60 debug emulator.

To run the application at the command prompt type:

```
epoc
```



Figure 2: The Series 60 emulator

Series 60 debug emulator will start up and Series 60 system shell will be shown as in Figure 2. Navigate to the “Other” folder using either the cursor keys on the emulator fascia or the PC cursor keys. Click on the action button in the middle of the cursor controls button to open the folder. Navigate to the HelloWorld application and click on the action button again to invoke the application.

5.2.2 Build and Run from the IDE

Normally projects such as the HelloWorld application are built and run from within the Microsoft Visual C++ 6.0 IDE as follows:

If the ABLD.BAT file does not already exist (or if the .mmp file or bld.inf file has changed), the build command file must be generated by typing:

```
bldmake bldfiles
```

followed by:

```
abld makefile vc6
```

This will build the project and workspace files (.dsp and .dsw) for Visual C++. They will be located under the \epoc32\build sub-folder structure e.g.,

```
\epoc32\build\sybian\6.1\Series60Ex\helloworld\helloworld\wins
```

By opening the workspace file HelloWorld.dsw in Visual C++ the application can be built within the IDE by pressing F7 and then run using Ctrl+F5. When asked for the executable, navigate to Epoc.exe in the folder \Epoc32\Release\wins\udeb in the SDK root. This launches the debug emulator; this is the default for development projects.

Series 60 debug emulator will start up and Series 60 system shell will be shown as in Figure 2. Navigate to the “Other” folder using either the cursor keys on the emulator fascia or the PC cursor keys. Click on the action button in the middle of the cursor controls button to open the folder. Navigate to the HelloWorld application and click on the action button again to invoke the application.

Alternatively, to run the application through the debugger, press F5. Ignore the warning that there is no debug information associated with Epoc.exe. It is the application (essentially a DLL) that will be debugged, not the emulator itself. Check the box so that the warning is not displayed again for this project. The app, and the associated debug information, are located in `Epoc32\Release\wins\udeb\z\system\apps\HelloWorld`.

5.3 Application UI Elements

Referring to Figure 3, the Status Pane is the solid (blue) 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 including the soft key labels.



Figure 3: The "Hello World" application

The status pane displays status information for the current application and state as well as general information about the device status, such as the signal strength and battery charging status. It occupies the top part of the screen. The status pane may not be visible in some applications or situations, such as games.

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, would be fully occupied by an application's view.

The control pane occupies the bottom part of the screen and displays the labels associated with the two soft keys.

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

5.4 The Symbian OS Application Framework

UIKON and Standard EIKON are key parts of the application framework. They not only provide the framework for launching applications, but also a rich array of standard control components (e.g., dialog boxes, number editors, date editors, etc.) that applications use at runtime. A typical application written for Symbian OS actually consists of four distinct components, each with a corresponding class within the UIKON/EIKON framework. They are:

- The application shell – is derived from **CEikApplication**. This class is instantiated first by the framework. Once created, it has responsibility for initializing the rest of the code. The new **CEikApplication**-derived class then creates...
- The document – is derived from **CEikDocument**. All applications have a **CEikDocument**-derived class and by default **CEikDocument** will create a default document file the first time an application is run. However, not all applications are file based. That is, they may not need to offer the user the ability to create, open, or edit documents. In such non-file-based applications, such as the Telephone application, the instance of the document class is little more than a shell required by the framework to create an instance of the **AppUi** class and typically a model/engine instance. In file-based applications the document class also manages the saving and restoring of data from persistent storage.
- The application UI - is derived from the Uikon class **CEikAppUi**. This class provides major functionality to all applications such as event handling, control creation, access to many useful system calls, etc. The **CEikAppUi**-derived class is typically responsible for creating one or more application views.
- The view – provides what the user actually sees on the screen. All applications have one default view; more complex ones, such as Calendar can offer many views. The view can be used simply to display data (exemplified in HelloWorld) or to collect data from the user in more interactive applications. For example, in many data-entry applications the data editors are simply standard controls provided by Uikon contained within the view. In the majority of applications the view(s) are derived from **CCoeControl** i.e. they are controls in their own right.

5.5 Series 60 Application Framework

Avkon is a UI layer specific to Series 60 Platform. It provides a wide range of user-interface components and implements a number of classes derived from the UIKON and Standard EIKON framework base classes that give Avkon applications properties and behavior characteristic of Series 60 Platform.

- **CAknDocument** – This class is provided as a base class for application documents and is derived from **CEikDocument**. By using this class, access to a default application document file is never initiated. This situation is acceptable to the majority of Avkon applications. Avkon disables document file creation by default when **CAknDocument** class is used as base class for application document.

- **CAknAppUi** – Avkon applications (except view architecture applications) derive from this class. This class supports several Avkon-specific functionalities:
 - KeySound support
 - Accessories for CBA and StatusPane
 - TextResolver – Avkon-specific error reporting from **CAknAppUi::HandleError()**
 - Avkon view architecture integration
 - Control dumping – Debug feature

CAknViewAppUi

All applications based on the view architecture (see Alternative GUI Designs below) must derive from this class, which is derived from **CAknAppUi**. The application views may be derived from **CAknView**.

- Application Startup – A class, **CAknApplication** is supplied that derives from **CEikApplication**. This class modifies **CEikApplication** by re-implementing **PreDocConstructL()** and **OpenIniFileLC(RFs& aFs)**. **PreDocConstructL** is implemented to ensure that an instance of the application being constructed is not already present. If it is present, then the application switches to the existing instance and then exits. This check is only carried out for non-embedded applications. By default, **.ini** files are not supported by Series 60 applications. **OpenIniFileLC()** is over-riden with a simple implementation that merely leaves if called. If an **.ini** file is to be used, the application is forced to implement this method in the application class to call **CEikApplication::OpenIniFileLC**.

5.6 Alternative GUI Designs

Application UIs can be simple with only one main screen, such as a calculator, or complex with many screens, e.g., a messaging application. Hence, three architectural approaches have been identified for writing an application GUI:

- Traditional Symbian OS Control Architecture - **CCoeControl** derived view(s)
- Dialog-based Architecture – views are all derived from Dialog classes
- View Architecture – view switching where application views may be derived from **CAknView**

The choice of application architecture will depend on the application complexity, the view navigation and communications requirements and the screen layout requirements. Whichever architecture is used, the top-level application UI class of each application will be derived from a single application UI base class. The base class does not enforce any choice of UI architecture that must be added by the developer.

Symbian OS applications are traditionally written using **CCoeControl** derived custom view controls placed on the applications control stack to act as application views. These controls can be created and destroyed, shown and hidden as required by the application to provide the appropriate behavior. This approach is very well suited to applications in Avkon. Since many of the Avkon applications will be based on existing UIs written using traditional methods, it is appropriate to follow the same style with Avkon.

5.7 UI Style

The physical appearance of an application's views, menus, dialog boxes, etc., can be made more professional by complying with the *Developer's Guide to Series 60 UI Category*, available as part of Series SDK documentation.

5.8 Series 60 Application Wizard

Developers may wish to use the console HelloWorld application as the basis of a test harness, or use the GUI version as the starting point for applications.

Alternatively, Series 60 AppWizard, included with the Series 60 SDK, provides developers with a simple and convenient way to generate a basic GUI application project from within Visual Studio. The application wizard can produce projects that conform to any of the three UI architectures described above.

The AppWizard will generate:

- Stub code and declarations for the four basic classes (App, AppUI, Document and View) associated with most Symbian OS applications
- All the build and project files necessary to build the project
- Simple resource files, and bitmaps that are used as default application icons
- The files needed to install the application onto a device

Use of the AppWizard is described further in the Series 60 SDK documentation in "Series 60 App Wizard Installation & User's Guide."

5.9 GUI Application Project Files

Developers must produce Component Definition (bld.inf) and a Project Definition (.mmp) files for each development project. A project must have a bld.inf file and will refer out to one or more .mmp files, i.e. one for each component. In simple projects there will be a single component e.g., a reference to the .mmp file for the application. In more complex projects that involve other components in addition to the application, such as DLLs, there may be multiple .mmp files. The component definition and project definition files are used by the tool chain to construct a build file (ABLD.BAT). This file is then used for various purposes e.g., creating other project and workspace files for the Visual C++ development environment, or for building the project for the emulator or target device in debug or release formats.

See Series 60 SDK documentation for more detailed descriptions of the syntax of bld.inf and .mmp files.

5.9.1 bld.inf

PRJ_MMPFILES

```
// Specify the .mmp files required for building the important component
```

```
// releasables.
```

```
\Symbian\6.1\Series60\Series60Ex\HelloWorld\group\HelloWorld.mmp
```

The example component definition file (bld.inf) above refers to a single project definition file i.e. HelloWorld.mmp. For a simple application or single component this is typically all there is in a bld.inf file.

5.9.2 HelloWorld.mmp

Key sections of the HelloWorld.mmp file are shown below and brief explanations of the key statements is provided. A project definition (.mmp) file specifies the properties of a project in a platform and compiler-independent way; it may then be used by the SDK build tools (abld.bat) to produce specific makefiles for particular platforms.

```
TARGET HelloWorld.app

TARGETTYPE app

UID 0x100039CE 0X10008ACE

TARGETPATH \system\apps\HelloWorld

SOURCEPATH ..\src

SOURCE HelloWorldApp.cpp
SOURCE HelloWorldAppUi.cpp
SOURCE HelloWorldDocument.cpp
SOURCE HelloWorldContainer.cpp

RESOURCE ..\data\HelloWorld.rss

USERINCLUDE . ..\inc

SYSTEMINCLUDE . \epoc32\include

LIBRARY euser.lib apparc.lib cone.lib eikcore.lib

LIBRARY eikcoctl.lib avkon.lib

#ifdef MARM

LIBRARY aknnotify.lib

#endif

AIF HelloWorld.aif ..\aif HelloWorldaif.rss c8
context_pane_icon.bmp context_pane_icon_mask.bmp list_icon.bmp
list_icon_mask.bmp
```

- TARGET is the name of the application
- TARGETTYPE is app, meaning a GUI application
- TARGETPATH is the location of the application and its components – this is always under \system\apps\
- The UID line specifies a unique system identifier for a GUI application — 0x100039CE and 0X10008ACE for the application itself

- The **SOURCE**, **SOURCEPATH**, **USERINCLUDE**, **SYSTEMINCLUDE** statements refer to the source files for the project with path information for source plus application and system header files
- The **RESOURCE** statement refers to source files that defines the majority of the user interface elements such as menus, dialogs, text strings etc
- The **LIBRARY** statement lists the application framework and graphics libraries required for linking a GUI application. Statements such **"#ifdef MARM"** statement can specify a conditional build, so that, in this case, the **aknnotify.lib** is used only if the project is built for a Series 60 target device
- The **AIF** statement refers to an Application Information File that contains icons and other application properties defined in the application resource file– more on this in section 6.2.1 later

5.10 Resource Files

Resource files (e.g., **HelloWorld.rss**) are used in Symbian OS to define the way a GUI application looks on the screen. Much of the information that defines the appearance, behavior and functionality of an application is stored in a resource file; external to the main body of the program, everything from the status pane, the menus, and the hotkeys, through to individual dialog boxes can be defined in the resource file. Individual resources are loaded very efficiently as required at runtime and hence the memory requirements are minimized.

Application resources are defined in text script files (typically with **.rss** extensions), they are compiled and compressed at build time into binary files that are used at run-time (by default with **.rsc** extensions). Resource files can be localized without recompiling the main program. For ease of localization, all user interface text is usually separated out in to a separate header file (by convention with a **.loc** extension) that is **#included** into the main resource file. It is the **.loc** files that are sent for translation into different languages.

Resources initially appear complex, but taken step-by-step they are quite straightforward. Comprehensive explanations and examples of application resource files are available in the Series 60 SDK.

5.11 Source Files

The following files make up the body of the application source.

HelloWorldApp.cpp	- The application
HelloWorldAppUi.cpp	- The application UI
HelloWorldDocument.cpp	- The document
HelloWorldContainer.cpp	- The view

HelloWorld.rss – standard resource file containing many UI definitions. This file is used by the resource compiler that is automatically called before the compilation of the source files, (only if the resource file has been updated since the last running of the resource compiler). The output of the resource compiler is typically a binary file that is used at runtime to supply the resource information as required e.g., **HelloWorld.rsc**.

HelloWorld_caption.rss – contains captions for the application's icon i.e. the application name for display in the “app launcher” of Series 60 UI. Two sizes are quoted to accommodate zoom levels of the UI. This file is also processed by the resource compiler during a full build.

Hello.uid.cpp – UID source file, automatically generated by the build tools e.g., under a folder such as `\epoc32\build\symbian\6.1\series60\helloworld\group\helloworld\wins`. Developers should never edit this file.

5.12 Header Files

HelloWorld.hrh – contains definitions of some UI control enumerated constants that are used in both source files (.cpp) and resource files (.rss). These constants are Command IDs, Key IDs or View IDs specific to an application. They may be used for key handling or for command handling in dialogs. For the moment the example HelloWorld application has three:

EExampleItem0

EExampleItem1

EExampleItem2

Note that `avkon.hrh` (located in `\Epoc32\Include`) contains many enumerated constants defined by Series 60 UI framework, this is a very important file for Series 60 applications. For example, one of the constants defined is `EAKnCmdExit`; an application's menu option Exit should always generate this command. The standard value passed by the framework when an application should close down (e.g., for a system backup) is `EEikCmdExit` as defined in `uikon.hrh`.

<code>HelloWorldApp.h</code>	- Header file for <code>HelloWorldApp.cpp</code>
<code>HelloWorldAppUi.h</code>	- Header file for <code>HelloWorldAppUi.cpp</code>
<code>HelloWorldContainer.h</code>	- Header file for <code>HelloWorldContainer.cpp</code>
<code>HelloWorldDocument.h</code>	- Header file for <code>HelloWorldDocument.cpp</code>
<code>HelloWorld.loc</code>	- Header file that is included in the .rss file containing user interface text strings etc. to make localization easier.

6. Executable Files and Locations

This section describes the executable files produced by the build process, i.e., the type and purpose of each file and its locations on an emulator and a target device.

6.1 HelloWorld Console Application Files

Under Series 60 emulator the HelloWorld console application consists of a single executable file, i.e., **HelloWorld.exe**. It is generated in the **\Epoc32\Release\wins\UDEB** folder for debug builds and the **\Epoc32\Release\wins\UREL** folder for release builds (emulator release builds are rarely done).

Under the emulator, these locations are essential for the correct execution of .exe files. On target hardware .exe type executables are conventionally located under a **\system\programs** folder but they will run from any location. However, it is possible that the target device's Messaging Inbox will recognize the Symbian OS executable file format in an incoming message attachment, but not open message and its attachments for security reasons.

6.2 HelloWorld GUI Application Files

At least two files make up a minimum GUI application, the .app and .rsc files (e.g., **HelloWorld.app** and **HelloWorld.rsc**). The **HelloWorld.app** file is the application executable, i.e., the output from the compilation and linking process; the **HelloWorld.rsc** file is the binary resource file produced by the resource compiler. Usually an application information (.aif) file is also supplied containing icons, captions in the supported languages and application capabilities such as whether embedding is supported, etc. An application will still function if the .aif file is not present, but it will have a default icon provided by the system and other application characteristics will be set by the system to defaults. If a caption resource file (**HelloWorld_caption.rsc**) was specified then an additional file for installation **HelloWorld_caption.rsc** will have been generated. See the 'Captions' section under 'AIF Files' later.

In Symbian OS, a GUI application is a special form of dynamic link library (DLL) that always has an .app file name extension. A special framework is provided by the system to load and correctly initialize GUI applications. Developers have to conform to the requirements of this framework and must provide implementations of a number of pure virtual functions in order that their application will start up correctly. Optionally implementations of a number of other virtual functions may be provided in order to achieve the desired specific behavior required by the developer.

In order to be automatically recognized by the system all GUI applications must follow a location convention i.e. they must be under a folder such as: **\system\apps\appname**. So in the case of the HelloWorld example, the files described above should be located in a folder called **\system\apps\HelloWorld**. I.e. **HelloWorld.app**, **HelloWorld.rsc**, **HelloWorld_caption.rsc**, and **HelloWorld.aif**.

A number of others files may be essential for the correct functioning of a particular application e.g., there may be need for a compressed multi-bitmap (.mbm) file, a specific data file etc. It is usual to put such essential files in the same location as the key application files i.e., in **\system\apps\appname**.

6.2.1 AIF Files

Application information (.aif) files are used at runtime, and store data concerning an application including:

Icons in various sizes used by the system to represent the application

Captions in the supported languages

Capabilities, such as document embedding, new-file creation, if the applications is hidden or not, and MIME-type support priorities

Each application should own an application information file (or .aif file), which is used to contain a bitmap and captions associated with that application. Without an .aif file an application will use a default icon, and the application name as a caption (without the file name extension) and MIME types and embedding will not be supported. It can be created independently of the application by the AIF Builder tool provided with the Series 60 SDK. The AIF Builder tool saves the details of .aif definitions in files with an .aifb extension.

The composition of .aif files can also be specified manually in resource script files with a .rss filename extension e.g., **HelloWorldaif.rss** and compiled with the Aiftool utility. The Aiftool utility and the syntax of .aif script files are documented in the Series 60 SDK.

6.2.1.1 Icons

Icons are used to represent applications, and their associated files/documents, when they are embedded, or when they are shown on the application shell. Icons may be provided in a range of sizes; the most appropriate size for the current container zoom state will be displayed. Supplying a variety of sizes helps to ensure that an icon will not have to be dynamically scaled when it is drawn at a particular size — scaling small bitmaps generally results in a marked loss of quality.

AIF Builder can launch an icon designer facility that will generate the bitmaps and masks that make up the icon. AIF Icon Designer helps produce such icons in the required Symbian OS-specific bitmap file format, called the multi-bitmap file format (.mbm).

If an application requires different icon bitmaps for different languages, this can be achieved by producing multiple copies of the .aif file, each containing the correct bitmap(s), using Aiftool. Each .aif file produced must be localized by saving with the extension aXX where XX is the 2-digit language code associated with the appropriate language. The application framework (AppArc) software has been modified to attempt to load the aif file associated with the current language, selected by the user.

6.2.1.2 Captions

Avkon provides the possibility of associating a short caption with each application. By default this will be identical to the caption found in the .aif file. However, it is possible for application authors to generate a separate caption file for each language, containing a caption and shortcaption. The shortcaption is used in an application grid display, whereas the long caption is used in an application list. The caption file is generated in the same way as a normal resource file. The resource structure used to create this caption file is defined in **apcaptionfile.rh**.

6.2.1.3 MIME support

Multipurpose Internet Mail Extensions, MIMEs, define a file format for transferring non-textual data, such as graphics, audio and fax, over the Internet. A Symbian OS application may specify any MIME types that it supports in the .aif file, and the priority of support that each type is assigned.

For more information on .aif files see *Series 60 Tools, Designing Series 60 C++ Applications* and the Series 60 SDK documentation.

7. Building for a Series 60 Device

Building for a Series 60 target device is done at the command prompt by invoking the GCC cross-compiler to build the executables in a suitable ARM binary format. To build for the 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
```

This will cause the build system to produce an ARMI release build of the application for execution on a target device using the GCC tool chain. The steps include C++ compilation, linking, resource compilation and .aif production.

The executable and data files (**HelloWorld.app**, **HelloWorld.rsc**, **HelloWorld_caption.rsc** and **HelloWorld.aif**) will now be located in **..\epoc32\release\armi\urel**. For testing on a Series 60 device, these four files have to be transferred to a device and located in a folder called **\system\apps\HelloWorld**. See 'Application Installation', below for options for doing the file transfer.

For more details about other build types and options, refer to the Series 60 SDK documentation.

8. Implementing On a Series 60 Device

Options for file transfer, PC connectivity, and, hence, application installation, may vary depending on the specific hardware and software available for the Series 60 device being used.

8.1 SIS File Installation

The Symbian Installation System provides a simple and consistent user interface for installing applications, data, or configuration information to Symbian OS phones. Developers (or end users) install components, packaged in SIS (.sis) files. Three potential installation options exist:

- Installation through the invocation of an SIS file located on a PC with subsequent application installation on to the phone through an Infrared or Bluetooth session between the PC and the phone.
- By transfer of an SIS file through OBEX, Infrared or Bluetooth, from another device such as a PC, other Symbian OS phone or any OBEX-enabled device, with application installation using the 'Application Controller' on the phone.

- Alternatively, SIS files can be sent as e-mail attachments and application installation is then be available via the 'Application Controller' on the phone. However, it is possible that the target device's Messaging Inbox will recognize the Symbian OS executable file format in an incoming message attachment, but not open message and its attachments for security reasons.

After installation, a small "stub .sis file" remains on the phone to control the un-installation of the application if required later.

8.2 SIS file creation

SIS files are created by the Makesis tool from input (.pkg) files such as **HelloWorld.pkg**. A .pkg file is a text file containing installation information for applications or files. They can be created manually or produced by the 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. Manually created .pkg files can be imported into Sisar.

Note that within the .pkg file, it is a requirement that a Series 60 Platform identification code be included. This enables a built-in mechanism to issue a warning if a user attempts to install non-Series 60 software onto a Series 60 device. For a fuller explanation of this requirement and the implementation details see:

<http://www.forum.nokia.com/021/1,,,00.html?fsrParam=1%2D3&fileID=2433> - the relevant document is entitled "Series 60 Platform Identification Code".

9. Developer Resources

9.1 Forum Nokia

Forum Nokia is Nokia's support forum for developers using its technologies and is found at <http://www.forum.nokia.com/>.

The forum contains a wealth of information on Symbian OS for Series 60 Platform, as well as free SDKs. Access to these resources is available free to all registered developers. Resources also include discussion forums devoted to Symbian OS development. In order to gain full access it is necessary to register. However, there is no charge for registration or for participation in discussion forums devoted to Symbian OS development. The forum also contains numerous technical papers devoted to Symbian OS development, including papers devoted to C++ and general configuration issues; these can be found in the "Symbian" section.

The Nokia Knowledge Network provides free discussion board related to Symbian questions at: <http://nkn.forum.nokia.com/category.cfm?oid=4237926224025259>

The Forum Nokia Knowledgebase provides additional information about Symbian OS such as:

- Answers to frequently asked questions
- Useful code snippets illustrating Symbian OS techniques

Also published on Forum Nokia are defect reports and their corresponding solutions, under the Nokia Knowledge Network section.

9.2 Symbian Developer Network

Symbian Developer Network (Symbian DevNet) is a primary resource for access to the technical and commercial information and resources developers need to succeed in the wireless space. The Symbian Developer Network is Symbian's developer service organization, providing developers using Symbian OS with support and information to help them produce well-written and stable applications for Symbian OS devices.

The Symbian Developer Network provides both free public discussion forums at:

www.symbian.com/developer/support.html

Professional support forums for subscribing members at:

www.symbian.com/developer/prof/index.html.

The professional C++ forums are staffed by Symbian developer consultants.

The Symbian Developer Network Web site includes a technical library that can be found at www.symbian.com/developer/techlib/index.html

Symbian DevNet provides:

- Discussion forums, code resources, and tips about Symbian OS development

- A portal to other such resources and technical and commercial services on Symbian DevNet partner sites
- A newsletter on what's new and hot on the Symbian DevNet Web site and across all our partner sites
- The Symbian OS Knowledgebase that provides up-to-the-minute information about Symbian OS such as:
 - Answers to frequently asked questions
 - Defect reports and the corresponding solutions
 - Useful code snippets illustrating Symbian OS techniques
 - Other development information that is not available in the Software Development Kit documentation

9.3 Books

Professional Symbian Programming, M. Tasker et al, Wrox Press.

Symbian OS Communications Programming, M. Jipping, Symbian Press, Wiley.