symbian Academy

Tool Chain

Introduction Part 1b

Tool Chain

This lecture examines

- The Symbian OS tool chain and development environment

The emphasis is on providing a background and purpose of the tools

- Tools are only really learnt by use and experience

symbian Academy

Build Tools

- Understand the basic use of **`bldmake`**, **`bld.inf`** and **`abld.bat`**
- Understand the purpose and typical syntax of project definition (MMP) files
- Understand the role of Symbian OS resource and text localization files

Build Tools

To build a Symbian OS program two build files are required:

- The component description file (**`bld.inf`**)
- The project definition file (**`projectname.mmp`**)

Build Processing

Symbian OS has its own platform-independent build file format

- Used by **`bld.inf`**
- To specify how a program is built

The **`bldmake`** tool

- Processes the **`bld.inf`** component description file
- Which contains the associated project definition files
- To generate a batch file - **`abld.bat`**

Build Processing

The **`bldmake`** tool can be called with the following options:

- **`bldmake bldfiles`** generates **`abld.bat`** and associated **`.make`** files
- **`bldmake clean`** removes all files generated by **`bldmake bldfiles`**
- **`bldmake inf`** displays the basic **`bld.inf`** syntax
- **`bldmake plat`** displays a list of supported build platforms

Build Processing

Build platforms

- Represent the various target platforms and thus binary formats

When `abld.bat` is invoked

- For example - to build code
- The particular platform (emulator or hardware) is specified as an argument

The most commonly used build platforms are:

- **`WINSCW`** which creates x86-format binaries for running code on the Windows emulator
- **`GCCE`** or **`ARMV5`** which create binaries to run on phone hardware
  - Built with the GCCE and RVCT compilers respectively

Build Processing

When one of the platforms is specified

- As an argument to the **`abld.bat`** command
- The makefile for that platform is generated and executed

The syntax of **`bld.inf`** is straightforward

- Its main purpose is to list project definition files
- And files that the build tools must export to another location before a build takes place

In the simplest case

- **`bld.inf`** lists just one project definition file to be built
- Specified following the **`PRJ_MMPFILES`** keyword

`bld.inf` Keywords

More complex `bld.inf` files

- Can be made up of a number of sections

Using the following keywords:

`PRJ_TESTMMPFILES`

- Specifies one or more project definition files for test code
- Which can be built by invoking **`abld test build`** rather than **`abld build`**

`bld.inf` Keywords

`PRJ_EXPORTS`

- Lists a series of files to be copied from the project directory to another directory
- Usually somewhere under `\epoc32`
- The export can be initiated by calling `abld export`
- The export is performed automatically as part of the `abld build` command

`PRJ_TESTEXPORTS`

- Lists a series of files to be copied from the project directory to another directory
- Usually somewhere under `\epoc32`
- The copy can be initiated by calling `abld test export`
- Performed automatically as part of the `abld test build` command

`bld.inf` Keywords

`PRJ_PLATFORMS`

- Can be used to list the platforms that the component supports

If this is not specified

- The **`abld`** command uses the default set

Each of the keywords

- Can be specified multiple times in any order

In addition to these keywords

- Extension makefiles can be used for build tasks
- Which are not offered by the generated makefiles
- Such as invocation of specialized tools or conversion utilities

Build Processing

When a change is made to `bld.inf`

- For example - to add a new header file under `PRJ_EXPORTS`
- `bldmake bldfiles` must be called again
- To generate a new version of `abld.bat`
- Plus any build makefiles that it uses

The `abld.bat` command

- Can be invoked from the command line with various arguments

The most commonly used are as follows ...

`abld.bat` Command Line Arguments

`abld build`

- Combines a number of other arguments
- **`export`**, **`makefile`**, **`library`**, **`resource`**, **`target`** and **`final`**
- To build the components specified as MMP files under the **`PRJ_MMP-FILES`** specifier in the **`bld.inf`** file

`abld test build`

- builds those components specified under **`PRJ_TESTMMPFILES`** in **`bld.inf`**

`abld.bat` Command Line Arguments

`abld makefile`

- Using the Symbian OS **`makmake`** tool
- Creates the makefiles for each project specified in **`bld.inf`**
- The makefiles are then used by **`abld`** to carry out the various stages of building the component
- This command is called each time a component is built
- The makefiles are always re-created
- Regardless of whether the corresponding MMP files have been changed or not since their last creation

`abld.bat` Command Line Arguments

`abld freeze`

- Freezes new DLL exports into `.def` files

`abld clean`

- Erases all the files created by a corresponding `abld` target command
- All the intermediate files created during compilation
- All the executables and import libraries created by the linker

`abld reallyclean`

- Does what `abld clean` does
- Also removes files exported by `abld export`
- And makefiles generated by `abld makefile` or the `abld test` equivalents

`abld.bat` Command Line Arguments

Each command

- Can be invoked on the projects specified under **`PRJ_MMPFILES`** in **`bld.inf`**
- By using **`abld XXX`** where **`XXX`** is the command
- On test projects specified under **`PRJ_TESTMMPFILES`**
- By using **`abld test XXX`**

Project Definition Files and MMP File Syntax

A project definition file

- Is a text file which gives the details needed to build a project
- Usually referred to as a project's MMP file

These include

- The project's source files
- The import libraries
- The locations of files included through use of **`#include`** preprocessor directives

Each statement in a project definition file starts with a keyword ...

Project Definition Files and MMP File Syntax

The main keywords will be described for a typical MMP file as follows:

```
TARGET          ASDExample.exe
TARGETTYPE      exe
UID             0 0xF1101100
CAPABILITY      NONE
SOURCEPATH      ..\src
SOURCE          ASDExampleAppUi.cpp
SOURCE          ASDExampleDocument.cpp
SOURCE          ASDExampleApplication.cpp
SOURCE          ASDExampleView.cpp
SYSTEMINCLUDE   \epoc32\include
USERINCLUDE     ..\inc
SOURCEPATH      ..\data
START RESOURCE  ASDExample.rss
TARGETPATH      \resource\apps
HEADER
END
START RESOURCE  ASDExample_reg.rss
TARGETPATH      \private\10003A3F\apps
END
// Generic Symbian OS libraries
LIBRARY euser.lib efsrv.lib ... // Others omitted for clarity
```

Project Definition Files and MMP File Syntax

TARGET

- Specifies the name of the file that will be built - **ASDExample.exe**

TARGETTYPE

- Indicates the type of file to be built
- In this case an executable application (**EXE**)

The most commonly used Symbian OS target types are

- **DLL**
- **EXE**
- **PLUGIN** (ECOM plug-in)

Project Definition Files and MMP File Syntax

Other supported **TARGETTYPE**s include

- **PDD** - physical device driver
- **LDD** - logical device driver
- **LIB** - a static library whose binary code is included directly in any component that links against it
- **EXEXP** - an executable which exports functions

EPOCEXE and **EXEDLL**

- Are sometimes used in Symbian OS EKA1
- Are no longer necessary in EKA2 because of its improved process emulation

Project Definition Files and MMP File Syntax

`UID`

- Specifies the final two of the target's three Unique Identifiers to identify the component

The target will have three UIDs

- The first value (UID1) does not need to be given

  It is automatically applied by the build tools according to the **`TARGETTYPE`**

No two executables

- May have the same UID3 value
- Values must be requested from Symbian Signed
- Allocated them from a central database

Project Definition Files and MMP File Syntax

`SECUREID`

- Is an optional keyword which is not used in the example
- Used to define the Secure Identifier (SID) for an executable
- Used to identify it

The SID

- Can be specified by a SECUREID statement in the project's MMP file
- If it is not specified the UID3 value is used instead

In MMP files where UID3 is not specified

- **`KNullUID`** (=0) is used as both the SID and UID3 value

Project Definition Files and MMP File Syntax

`VENDORID`

- Is an optional keyword (not used in the example given)
- New in Symbian OS v9.1

An EXE

- May contain a vendor ID (VID)
- Specified by the **`VENDORID`** keyword
- The use of a VID identifies the supplier of the binary
- Its use is not mandatory

Project Definition Files and MMP File Syntax

CAPABILITY

- A new keyword in Symbian OS v9.1
- Used to restrict the use of certain sensitive system APIs
- To callers with a particular level of privilege

The capabilities assigned to an executable

- Listed following the **CAPABILITY** keyword

If the **CAPABILITY** keyword is not used

- The capabilities assigned to the binary default to **CAPABILITY NONE**

Project Definition Files and MMP File Syntax

The maximum set of capabilities

- That can be used can be specified by **`CAPABILITY ALL`**
- Very few components are built with this level of privilege

In general

- For code which has a high level of privilege
- The maximum set of capabilities specified will be **`CAPABILITY ALL -TCB`**

Project Definition Files and MMP File Syntax

SOURCEPATH

- Specifies the location of the source or resource files listed in the **SOURCE** declaration
- Can be a relative location
- Or a fully qualified path

The keyword

- Can be used multiple times to specify different directories
- Can be omitted entirely if all source files are in the same directory as the MMP file.

Project Definition Files and MMP File Syntax

SYSTEMINCLUDE

- Specifies the directory in which files included in the code
- Using **#include <>** can be found

All global headers

- Should be stored in **\epoc32\include**
- or a subdirectory thereof

Project Definition Files and MMP File Syntax

USERINCLUDE

- Specifies the directory in which files included in code
- Using #include "" can be found.
- Can be a relative path
- or a fully qualified path

Directories specified with **USERINCLUDE**

- Are only one of three locations that may be searched for header files

The two other directories:

- The source file directory
- The **SYSTEMINCLUDE** directory.

Project Definition Files and MMP File Syntax

`START RESOURCE...END`

- Specifies a resource file
- which contains text and specifications of user interface elements

These keywords

- Replace the use of **`RESOURCE`** statements

  (that were used in MMP files in versions of Symbian OS earlier than v9.1)

An application may have several resource files

- Each is specified separately in **`START RESOURCE`** ... **`END`** blocks

If the project has a GUI

- At least one of these resource files will be needed

Project Definition Files and MMP File Syntax

START RESOURCE

- Indicates the beginning of a block of information about an application resource file

The resource file

- Should be the same directory as the MMP file
- Or in a directory specified by a preceding **SOURCEPATH** declaration

END

- Indicates the end of the resource file information block

Project Definition Files and MMP File Syntax

In the example

- The second block specifies a registration resource file for the ASDExample application

```
START RESOURCE          ASDExample_reg.rss
TARGETPATH              \private\10003A3F\apps
END
```

This file contains

- Non-localizable information required by the application launcher

Project Definition Files and MMP File Syntax

Such as

- The application's name
- UID and properties
- And other information used by the launcher

For example

- The application's caption the name displayed for the application in the system shell
- And its icons are defined separately so that they can be localized

The location of these definitions

- Is provided in the registration file

Project Definition Files and MMP File Syntax

`TARGETPATH`

- Specifies the build location for a compiled resource (**`.rsc`**) as described later in this section

In the example

- The **`ASDExample.rss`** resource is compiled
- To generate output in the **`\resource\apps`** directory
- This is the standard location for compiled resource files

The second resource

- The registration file is built to **`\private\10003a3f\apps`**
- Which is the standard location for registration information

Project Definition Files and MMP File Syntax

The build location

- For binaries resulting from compilation of C++ code
- Used to be specified using the **`TARGETPATH`** keyword

However

- In the secure platform of Symbian OS v9.1
- All executable code must run from the phone's **`\sys\bin`** directory

  (This is covered more in a later lecture on Platform Security)
- The **`TARGETPATH`** keyword is thus now redundant
- Except to build resource files to their appropriate locations

Project Definition Files and MMP File Syntax

HEADER

- Is an optional keyword
- Causes a resource header file (**.rsg**) to be created in the system include directory (**\epoc32\include**)
- This allows the C++ code to use the names of specific resources
- Defined in the associated resource file

In the example

- A header file is generated for access to the resources
- Specified in **ASDExample.rss**
- No resource header file is generated for the registration resource file

Project Definition Files and MMP File Syntax

`LIBRARY`

- Lists the import libraries needed by the application
- No path needs to be given
- Each library statement may contain several libraries separated by a space
- More than one **`LIBRARY`** declaration may also be used

Project Definition Files and MMP File Syntax

`EPOCSTACKSIZE`

- Keyword can be used to increase the stack size
- To the following decimal or hexadecimal value

This option should be used with care

- Allocating extra stack space to one application
- Reduces the available space for others

If an application demands a large stack

- It should be analyzed for potential improvements and optimizations

Project Definition Files and MMP File Syntax

`EPOCHEAPSIZE`

- Is an optional keyword - is not used in the example
- It can be used to specify the minimum and maximum sizes of the initial heap for a process
- Either as decimal or hexadecimal values
- The default sizes are 4 KB minimum and 1 MB maximum

The minimum size

- Specifies the RAM that is initially mapped for the heap's use
- The process can then obtain more heap memory on demand until the maximum value is reached
- The values specified are rounded up to a multiple of the page size (4 KB)

Project Definition Files and MMP File Syntax

`EXPORTUNFROZEN`

- Is an optional keyword
- Used by DLLs that are not frozen to have complete `.def` files
- The `.lib` import library is created and all exported functions
- Including unfrozen ones appear in the import library

Resource Files

Resource files

- Are typically used to specify the user interface elements of a GUI application
- Such as the menu bars and dialogs
- They can also be used for any application type

Resource files are also used to define:

- The behavior and functionality of a Symbian OS application
- The application properties that are used by the application launcher
- Other literal strings and constant data used in the application
- e.g. dialog text and error messages

Resource Files

To maintain platform independence for a range of hardware

- The resource specifications are kept separate from the executable for each target platform

Resources are specified

- In a human-readable text file
- Compiled independently with the Symbian OS resource compiler into a separate binary

This separation

- Reduces the effort required to move applications between different hardware platforms

Resource Files

The syntax used for resource specification

- Provides good support for localization
- By allowing a separation of text from graphics

This not only facilitates translation

- But also allows a multi-lingual application to be created
- Without recompilation of the main application code
- The application is supplied as a single executable
- Together with a number of language-specific resource binaries

Resource Files

Resource files

- Are written as text files in a Symbian OS-specific syntax

They are then compiled on their own

- Using the command-line resource builder tool **`epocrc`**
- First passing the resource file through the C++ pre-processor
- Compiles it with the Symbian OS resource compiler **`rcomp`**

Or as part of the standard build tool chain

- From the command-line or within an IDE

Resource Files

Resource files

- Contain elements which begin with one of the Symbian OS resource keywords

Such as

- **`RESOURCE`**
- **`STRUCT`**
- **`ENUM`**

The resource file is named with the extension **`.rss`**

- When the resource compiler is invoked on the **`.rss`** file
- It generates two outputs ...

Resource Files

The resource compiler generates two outputs

- The binary resource file as a **`.rsc`** file
- a **`.rsg`** header file in **`\epoc32\include`**
- Which is built if the MMP file specifies the **`HEADER`** keyword in the **`START RESOURCE ... END`** block

The **`.rsg`** file

- Contains **`#define`** statements for each resource defined in the **`.rss`** file
- The header can be used by C++ application code
- To access elements in the resource binary
- By including it using the **`#include`** pre-processor directive

Resource Files

A localization file is a text file typically named with

- A `.rls` extension for UIQ
- A `.loc` extension for S60
- Included directly in the resource file

Other Symbian OS header file types

- That can be used in a resource file include

`.hrh`

- A header file that can be shared between C++ and resource files

`.rh`

- A header file used purely by a resource file

symbian Academy

Hardware Builds

- Understand that the ARM C++ EABI is an industry standard optimized for embedded application development
- Recognize basic information about the RVCT and GCCE compilers, which can be used for target hardware builds
- Understand that ARMV5 supports both 32-bit ARM and 16-bit THUMB instructions, and appreciate the difference with respect to speed and size

The EABI Standard

Symbian OS natively runs on ARM processors

- Code must be built with a compiler supporting the Embedded Application Binary Interface (EABI)
- This is a standard for the interfaces of binary code running in ARM environments
- It allows the inter-operation of binaries produced by different compilers that conform to the standard

The standard was designed

- To give efficient memory usage and data access time
- Interoperability between different compiler vendors

The EABI Standard

The Symbian OS build tools define native build targets that invoke either:

- A suitable version of the GNU Compiler Collection (GCC), for which the target is identified by the Symbian OS build tools as GCCE
- ARM's RealView Compiler Tools RVCT 2.2, for which the target is identified by the Symbian OS build tools as ARMV5

RVCT is intended for

- Symbian licensees such Nokia and Sony Ericsson to build ROMs for their handset products

GCCE is intended for the majority of Symbian developers

- It is delivered with any SDK for phone products based on Symbian OS v9.1
- It is also available for free download on the Internet

The GCCE Target Compiler

GCCE is a version of the Open Source GNU C++ compiler

- It is intended only for building applications
- It cannot be used to compile the full OS

GCCE can be used either

- From the command line
- Or invoked from within a development IDE such as CodeWarrior or Carbide.c++

The GCCE build target

- Uses the same **`.def`** file format as the ARMV5 target
- The tools default to looking for **`.def`** files in the project's **`\EABI`** directory

The GCCE Target Compiler

The GCCE compiler is very strict

- In checking the source code conforms to the ANSI C++ standard
- Some source code which previously compiled with less strict compilers (for example RVCT 2.1) may no longer compile.

The RVCT Target Compiler

RVCT is used by Symbian to compile Symbian OS

- And by its licensees to develop ROM-based code
- It gives the best performance and smallest code compared to other alternatives
- It must be purchased separately

Like GCCE

- RVCT can be used either from the command line
- Or invoked from within a development IDE such as CodeWarrior or Carbide.c++

ARM and THUMB

All current Symbian OS smartphones

- Are based on the ARM processor

Which has two instruction sets:

- A 32-bit set (known as ARM)
- A 16-bit set (called THUMB)
- Code compiled to one set can interoperate with the other

The ARM instruction set is fast

- Uses more memory per instruction

THUMB is more compact but slower

- That is more instructions are required to perform the same work

ARM and THUMB

The build tools apply this policy when building projects:

- Kernel-side code is built for ARM
- while other code (user-side) is built for THUMB
- All code builds into the same ARMV5 subdirectory of `\epoc32\release\`

ARM and THUMB

There are a number of ways to override the policy and to build user-side code for ARM:

- In the **`bld.inf`** file the **`BUILD_AS_ARM`** qualifier can be used to instruct an ARMV5 build not to build the project for the THUMB instruction set
- But explicitly for the ARM instruction set

```
PRJ_MMPFILES
ASDExample.mmp BUILD_AS_ARM
```

- To specify that a project should always be built as ARM in an MMP file
- The keyword **`ALWAYS_BUILD_AS_ARM`** can be specified

symbian Academy

Installing an Application to Phone Hardware

- Recognize the package file format used for creation of SIS installation files

Installing an Application to Phone Hardware

Unlike the Windows emulator

- Where binaries can simply be copied for testing

The only way to deploy code onto phone hardware

- Is for the software installer to read it from an installation package or SIS file (`.sis` extension)

To create a SIS file

- A package file (`.pkg`) is used to specify the files and metadata associated with an application
- Which is passed to the SIS file creation tool (`MakeSIS`)

The package file contains

- A list of the files, rules, options and dependencies required for the application

Installing an Application to Phone Hardware

The PC-based MakeSIS tool

- Reads the .pkg package file and generates a SIS installation

The SIS file contains

- All the information necessary for the Symbian OS software installer to install an application to the phone
- Except for the digital signature

Not most handset manufacturers

- Will require installation packages to be digitally signed before the application contained can be installed onto the phone

Installing an Application to Phone Hardware

The following is an example of a package file for the **`ASDExample`** application

- In a **`.pkg`** file lines preceded by semi-colons are comments and blank lines are ignored

```
; ASDExample.pkg
; Languages - English and French
&EN, FR
; List of localized vendor names
%{"SymbianPress", "SymbianPress"}
; The non-localized, globally unique vendor name
:"SymbianPress"
; Package header
#{"ASDExample"}, {"ASDExample"}, (0xF1101100), 1, 0, 0, TYPE=SA
; ProductID for UIQ 3.0
[0x101F6300], 3, 0, 0, {"UIQ30ProductID"}
```

Continued ...

Installing an Application to Phone Hardware

```
; Files to install for my directory application
; Paths are relative or fully qualified
{"english_info.txt" "french_info.txt"} -
"!:\Documents\ASDExampleGuide.txt"
"..\epoc32\release\gcce\urel\ASDExample.exe"-
"!:\sys\bin\ASDExample.exe"
"..\epoc32\data\Z\Resource\Apps\ASDExample.rsc"-
"!:\Resource\Apps\ASDExample.rsc"
"..\epoc32\data\z\Private\10003a3f\Apps\ASDExample_reg.rsc"-
"!:\private\10003a3f\import\apps\ASDExample_reg.rsc"
IF tkeyboard=1 ; phone has keypad only
"keypad_shortcut_config.txt"-"!:\private\F1101100\shortcut.txt"
ELSEIF tkeyboard=2 ; phone has full QWERTY keyboard
"keyboard_shortcut_config.txt"-"!:\private\F1101100\shortcut.txt"
ELSE ; Display a "No shortcuts are available for this phone" message
    "noshortcut.txt"-"", FILETEXT, TEXTCONTINUE
ENDIF
"readme.txt"-"", FILETEXT, TEXTCONTINUE
```

Installing an Application to Phone Hardware

The line preceded by **&** is the languages section

- It lists the supported language variants for the application
- Using two-character codes as set out in the Languages Table of the Symbian OS Library

In the example

- The **ASDExample.exe** application supports English (**EN**) and French (**FR**)

The sections preceded by % and :

- Are the localized and non-localized vendor names
- Localized vendor names are used in dialogs shown to the user
- While the non-localized vendor name is used internally by the software installer

Installing an Application to Phone Hardware

The line beginning with # is the package header

- This line provides the name of the application which is displayed in the installation dialogs
- The application's UID (as specified in the MMP file), version information and the installation package type

The package type indicates the type of installation

- Since different types have different rules on how files may be installed or uninstalled

The example uses **`TYPE=SISAPP`**

- Can also be specified using the abbreviation **`SA`**
- Or omitted entirely since it is the default
- This type identifies the component to be installed as an application
- Other types include a patch type and a partial upgrade type

Installing an Application to Phone Hardware

The line beginning with a hexadecimal UID in brackets is mandatory

- Is used to ensure that only applications designed and tested for specific phone hardware can be installed to it
- The important values are the hexadecimal UID (`0x101F6300`) and the string in quotes (`UIQ30ProductID`)

In the example

- These restrict the installation of `ASDExample.exe` to UIQ 3.0 phones
- The equivalent for a phone which runs on the S60 3rd Edition platform is `[0x101F7961], 0, 0, 0, {"Series60ProductID"}`

Installing an Application to Phone Hardware

Condition blocks (`IF ... ELSEIF ... ELSE ... END`)

- May be used to control aspects of the installation

In the example

- The condition block tests the `TKeyboard` attribute (from `HalData::TAttribute`) at install time
- Installing a shortcut configuration file according to whether the phone has a full QWERTY keyboard or a simple numeric keypad.

Installing an Application to Phone Hardware

If the `TKeyboard` attribute is neither of the values expected (=1 or =2)

- The contents of a text file (**`noshortcut.txt`**) is displayed to the user during the installation
- As it is not installed onto the phone, no destination location is specified for it

The instruction includes some options for the display:

- **`FILETEXT`** indicates to display the file during installation

  Other options include running an executable or creating a blank file in a specified location

- **`TEXTCONTINUE`** provides a continue button which will dismiss the text file and continue installation

  Other options include forcing the installation to exit or offering the user the opportunity to abort the installation

Installing an Application to Phone Hardware

These kinds of file display

- Can also be useful for showing basic information
- Such as a license agreement at installation time

The example

- Also uses it to show a **`readme.txt`** file during the installation of **`ASDExample.exe`**

The rest of the package file lists the files to install

- The filename before the hyphen indicates a file on the PC
- While the location after the hyphen is the destination path on the phone

Installing an Application to Phone Hardware

Specifying an exclamation mark

- In place of a drive letter in the target filename is recommended
- It means the user will see a dialog to give a choice of drive on which to install the application

If the drive were to be hard-coded

- The user may not have space available on that particular drive, which would make it impossible to install to it
- It is better to offer the user a choice of installation drive.

The line prefixed with { specifies a list of files

- Of which only one will be installed
- Depending on the language selected by the user during installation.

symbian Academy

The Symbian OS Emulator

- Understand the purpose of the Symbian OS emulator for Windows
- Recognize differences between running code on the emulator and on target hardware

The Symbian OS Emulator

The Symbian OS emulator is a Windows application

- Called **`EPOC.EXE`** which simulates phone hardware on the PC
- It is in effect a port of the Symbian OS kernel to the Win32 platform

The emulator enables Symbian OS

- Software development to be substantially PC-based in its early stages
- The final development stages will require the use of phone hardware.

The emulator runs in a single process

- Which means that on Windows each Symbian OS process is actually loaded as a DLL
- And runs inside a separate thread within the single Win32 emulator process

Reasons for Using the Symbian OS Emulator

The emulator saves time in the early stages of development

- Because a code development IDE such as CodeWarrior or Carbide.c++
- Can be used to debug the code and resolve most initial bugs and problems

For example if a panic occurs in the code

- The debugger can provide comprehensive information to diagnose the error condition that caused it

Reasons for Using the Symbian OS Emulator

For hardware testing

- An installation file must be created signed if necessary
- Transferred to the phone and installed

This can be time-consuming

- In the early phases of development when code changes are frequent
- The emulator does not need code to be formally installed
- Which makes the development process much faster

Reasons for Using the Symbian OS Emulator

For emulator builds the system also writes output to file

- Which can be inspected for information if a panic occurs
- To check for system warnings such as platform security violations

The file is located

- In the directory associated with the Windows **`TEMP`** environment variable
- Is named **`epocwind.out (\%TEMP%\epocwind.out)`**

Reasons for Using the Symbian OS Emulator

The emulator can be configured

- Through an initialization file called **`epoc.ini`**
- Is stored in **`\epoc32\data`** where all configuration files are located for the emulator

For normal use **`epoc.ini`**

- Does not need to be modified

But it can be used - for example

- To add customized virtual drives
- Change the size of the heap
- Or map areas of the emulator fascia to act as virtual keys

  Allowing emulation of phone hardware keys such as the navigation buttons

Differences Between the Emulator and Phone Hardware

The emulator programming environment

- Tends to be more forgiving than that for native code running on phone hardware
- For example ...
- Code which uses non-constant static variables will compile for the emulator, but will not compile for the ARM platform

Some code may run successfully on the emulator but fail on a real phone

- For example ...
- It is possible for one process to access the memory of another process on the Windows emulator without causing a memory exception
- On the hardware where memory protection is enforced by Symbian OS memory management the same code will generate a memory exception

Differences Between the Emulator and Phone Hardware

Symbian has tried to ensure that the Windows emulator

- Provides as faithful an emulation as possible of Symbian OS running on target hardware.
- The emulator has processes and thread scheduling that are almost identical to those on real Symbian OS phone hardware

However there are some differences

- For example the memory model for a real phone is different to that of the emulator

The underlying hardware is different

- It is not possible to use the same device driver and hardware abstraction layer code on both the emulator and a real phone

For this reason

- The emulator cannot be used for low-level programming such as that for device drivers

Differences Between the Emulator and Phone Hardware

Other differences between the emulator and hardware include the following:

Bootstrap:

- On real phones, the first Symbian OS program to run is a bootstrap program
- Which performs various hardware initialization tasks before starting the kernel
- The emulator does not need to perform these tasks and simply starts the kernel
- The remainder of the boot process is similar on the emulator to the native target

File system support:

- The emulator can emulate a range of file system and drive types
- But the performance and size of the emulated drives may not be exactly the same as expected for real hardware

Differences Between the Emulator and Phone Hardware

Floating-point behavior:

- Symbian OS provides access to IEEE-754 single-precision and double-precision floating-point values
- Through the types **`TReal32`** (C++ float type) and **`TReal64`** (C++ double type)
- The emulator is implemented on Intel x86 processors which have floating-point hardware - this support is used
- Target hardware may or may not have floating-point hardware support
- Where it does not - the calculations are performed in software
- Thus there may be significant performance differences between emulator and hardware versions of the code using floating-point arithmetic

Differences Between the Emulator and Phone Hardware

Serial ports:

- The emulator provides emulation of serial ports through Windows serial ports
- This is generally adequate for most purposes but may not provide the same performance as a real device
- Some applications have found that high latency times in Windows serial ports have caused some communications data to be dropped

Differences Between the Emulator and Phone Hardware

Timers:

- The standard timer resolution is 1/64th second on all platforms, including the emulator
- There is also a high-resolution timer accessed through methods such as **`User::AfterHighRes()`** and **`RTimer::HighRes()`**
- This has 1 ms resolution on reference hardware
- Defaults to 5 ms on the emulator
- But it can be changed by setting the timer period (in ms)
- Through the **`TimerResolution`** variable in the **`epoc.ini`** configuration file

Differences Between the Emulator and Phone Hardware

Machine word alignment:

- The 32-bit RISC architecture used by the phone hardware
- Requires that 32-bit quantities must be aligned to a 32-bit machine word boundary
- That is their address must be a multiple of four or an access violation will be generated
- This is not the case for code executing on the emulator which will run successfully

Differences Between the Emulator and Phone Hardware

Pixel sizes:

- There is a slight difference in the pixel sizes on the Windows emulator and on phone hardware
- Thus text and graphics may be displayed differently on the phone from the way it appears on the emulator

USB support:

- Symbian OS provides USB client support on phone hardware
- But there is no such support in the emulator

Emulator File System

The file system of the phone is mapped to the PC as follows:

- The internal writable drive (`c:`) is usually mapped to `\epoc32\ winscw\c`
- The ROM is mapped to `\epoc32\release\winscw\udeb\z` for the debug build
- And `\epoc32\release\winscw\udeb\z` for the release build

The emulator's configuration file can be modified

- To add other virtual drives if necessary
- The standard drives can be mapped to alternative locations as required

Emulator File System

There is one exception to the file system mapping

- Which is the location of the executables

On the phone

- All executables are stored in the **`\sys\bin`** directory

On the emulator

- The executables are loaded from where they are built
- That is the **`\epoc32\release\wins\udeb`** or **`\epoc32\release\wins\urel`** directory

Emulator File System

The emulator can also be set up

- So that it behaves as if a removable media card is present
- For example - a Memory Stick or MMC

This can be used to test how an application behaves

- When reading and writing data to the card
- Or when the card is removed and/or swapped

Emulator File System

It is possible to emulate

- A user opening and closing the removable media drive door
- Replacing and removing the media card
- Assigning a password to an emulated card

MMC emulation does not involve access to any kind of hardware interface

- Instead, the memory area of each emulated card is represented by a file
- A `.bin` type file in the Windows system temp directory

symbian Academy

Tool Chain

- ✓ Build Tools
- ✓ Hardware Builds
- ✓ Installing an Application to Phone Hardware
- ✓ The Symbian OS Emulator