

Selected topics in Symbian programming

Descriptors, hello-world, dialogs & active objects

Søren Debois

Department of Theoretical Computer Science
IT University of Copenhagen

September 23rd, 2005

Outline

- 1 Descriptors for Strings & Binary Data
- 2 Example minimal application (recap)
- 3 Dialogs
- 4 Active Objects

Representations of Strings (1)

By string, we mean a sequence of characters, e.g., this sentence, or the canonical 'Hello, World!'.

- In C and old-fashioned C++, we use null-terminated strings, i.e., sequences of bytes in which the last one is zero by convention. We must do our own memory management, which can be awkward. For example, appending `s` to `t` is done by

Example

```
char* u = new char[strlen(s) + strlen(t) + 1];  
strcpy(u, s);  
strcat(u, t);
```

Representations of Strings (2)

In Java and modern C++, we have nice `String` classes freeing us from having to worry about anything. We can now append without thinking about copying:

Example

```
String u = s + t;
```

In Symbian OS, we get a solution somewhere between the low-level zero-terminated strings, and the high-level `String` classes. We use **Descriptor Classes**, which contain basically **a pointer** and **a length**.

Strings and Memory

In C++, there are three places null-terminated strings can live.

- On the stack (automatically allocated).
- On the heap (dynamically allocated).
- In the program code (for constant strings). We say that the string is placed in read-only memory (ROM).

Allocating strings in C++

Here is how to allocate strings in these three different ways in C++.

Example

```
// ROM
const char* hello_rom = "Hello, world!";

void f () {
    // Stack
    char hello_stack[strlen(hello_rom) + 1];
    strcpy(hello_stack, hello_rom);

    // Heap
    char hello_heap* = new char[strlen(hello_rom) + 1];
    strcpy(hello_heap, hello_rom);

    // ROM
    printf("Hello, world!\n");
}
```

Allocating strings in Symbian (1)

... and here is how to do it in Symbian.

ROM

```
_LIT(KHelloRom, "hello");
```

(The “_LIT” is short for literal.) But `KHelloRom` is no longer a pointer! Instead, we can get to it using a wrapper class `TPtrC` that contains both the actual pointer *and the length* of the string.

Pointer

```
TPtrC helloPtr (KHelloRom);
```

`TPtrC` has a member function `Ptr()` that give us the actual pointer. (The `C` stands for constant. There is also a class `TPtr`.)

Allocating strings in Symbian (3)

To allocate a string on the stack, we use a `TBufC`.

Example

```
TBufC<5> helloStack(KHelloRom);
```

This object implements a 5-letter automatically allocated buffer. (Immutable, because it is a `TBufC`.) As `TPtrC` is “a pointer and a length”, so is `TBufC<5>` “some space on the stack and the length 5”. (`TBufC<5>` is a “template class”.)

Allocating strings in Symbian (3)

To allocate a string on the heap, we use a `HBufC`, that is, “some space on the heap and a length”.

Example

```
HBufC* helloHeap = KHelloRom().AllocLC();
```

What actually happens here is:

- 1 We cast `KHelloRom` into its base, `TDesC` by invoking its `operator()`.
- 2 `TDesC.AllocLC()` (1) allocates a new `HBufC` on the heap, (2) allocates space on the heap for whatever the `TDesC`'s contents, (3) copies these contents onto that space, and (4) pushes the new `HBufC` onto the cleanup stack.

Modifying C++ Strings

Recall the earlier example of appending strings.

Example

```
const char* s = "hello, ";  
const char* t = "world!";  
char* u = new char[strlen(s) + strlen(t) + 1];  
strcpy(u,s);  
strcat(u,t);
```

Modifying Symbian Strings (1)

Now in Symbian.

Example

```
_LIT(Ks, "hello, ");  
_LIT(Kt, "world!");  
TBuf<13> helloWorldStack(Ks);  
helloWorldStack.Append(Kt);
```

Modifying Symbian Strings (2)

TBuf is the descriptor that allocates its space on the stack. How do we manipulate HBufC? We don't. It's HBufC. There is no HBuf. We have to access it through a TPtr:

Example

```
HBufC* helloWorldHeap = HBufC::NewLC(Ks().Length() +
                                     Kt().Length());
TPtr helloWorldAppend(helloWorldHeap->Des());
helloWorldAppend = Ks;
helloWorldAppend.Append(Kt);
```

The TPtr keeps the HBuf synchronized: The used-length of the HBuf will grow when the TPtrs do in the last two statements. (Note the use of overloaded operator=.)

Descriptor Type Summary

These are the descriptor classes.

	Immutable	Modifiable
Pointer	TPtrC	TPtr
Buffer	TBufC<13>	TBuf<13>
Heap	HBufC	
Base	TDesC	TDes

References

- `TDes` and `TDesC` has members for all the operations one expects of strings (comparisons, substrings, searching, ...). Look them up in the documentation.
- Descriptors can also be used for binary data; lookup `TDes8` and `TDesC8`.
- Chapter 5 of “Symbian OS C++ for Mobile Phones” contains a comprehensive introduction to descriptors. (These slides borrow heavily from that chapter.)

Length of Strings

A 1-letter string takes up 2 bytes

(Because Symbian uses UTF-16 encoding of characters.)

Minimal application (recap)

A minimal UIQ application has four classes:

- An **application** (derived from `CApaApplication`). This class defines the properties of the application (say, the UID), and is responsible for constructing a document.
- A **document** (derived from `CApaDocument`). This class is the application's data model, and is responsible for creating a UI.
- A **UI** (derived from `CCoeAppUi`). This class generates and contains views, and handles **commands**.
- One or more **views** (derived from `CCoeControl`). This class handles display and low-level user-input.

(Besides C++-classes, it must also have a **project specification** file (`.mmp`), which contains all meta-information needed to build the application, and one or more **resource files** (`.rss`), which specifies static data, such as the layout of dialogs etc.)

Example minimal application (recap)

Let's look at an example application: The "Hello World" application from the SDK. Specifically,

- Symbian Series 60 C++ SDK 2.1
- Project Specification File:

```
C:\symbian\7.0s\Series60_v21_CW\Series60Ex  
  \helloworld\group\helloworld.mmp
```

(I need to manually add the file `helloworld.h` after importing `helloworld.mmp`, but my Metrowerks installation is partly broken. Your mileage may vary.)

What is a dialog?

Watch, as I start the emulator and click Configuration→Settings→Device settings to reach a representative dialog. We see that

- Dialogs always have controls stacked vertically. A control has a caption possibly some form of input field.
- Dialogs scroll when there are too many controls for the window.
- Dialogs can be subdivided in tabs, on the top.

Dialogs in Symbian are so simple that they are straightforward to program.

Example dialog

Let's look at an example dialog, a sample application doing encryption from the SDK. Specifically,

- Symbian Series 60 C++ SDK 2.1
- Source files

```
C:\symbian\7.0s\Series60_v21_CW\  
Series60Ex\cipher\src\cipherappui.cpp
```

- Project Specification file

```
C:\symbian\7.0s\Series60_v21_CW\  
Series60Ex\cipher\group\cipher.mmp
```

Active objects

- Active Objects is the Symbian C++ way to implement timers and, more generally, asynchronous execution, where you want something to happen when some event not covered by the main event loop occurs.
- Active objects derive from `CActive`
- They register themselves upon construction with the `CActiveScheduler`.
- They express interest in some happening by issuing calls to Symbian by asking, say, for a `RTimer` or `RSocket`.
- They the happening occurs, the active objects `RunL` method is implemented.
- If for some reason the happening can never occur, say, because the connection was lost, or the timer was cancelled by someone else, the `DoCancel` method of the active object is called.

Example active object

Let's look at an example application, a socket-reading example from the SDK. Specifically,

- Symbian Series 60 C++ SDK 2.1
- Source files

```
C:\symbian\7.0s\Series60_v21_CW\  
    Series60Ex\sockets\inc\socketsreader.h  
C:\symbian\7.0s\Series60_v21_CW\  
    Series60Ex\sockets\engine\socketsreader.c
```

- Project Specification file

```
C:\symbian\7.0s\Series60_v21_CW\  
    Series60Ex\sockets\group\sockets.mmp
```

Under the hood of active objects

But wait! The active object itself was never passed to anyone. All we ever did was pass our `iStatus` field in the socket call

```
iSocket.RecvOneOrMore(iBuffer,  
                      0,  
                      iStatus,  
                      iDummyLength);
```

What gives?

Our active object registered itself with the `CActiveScheduler` while under construction. This is enough, because the service provider (in this case the `RSocket`) communicates with the `CActiveScheduler`.

Under the hood of active objects (II)

Here is the sequence of events.

- 1 The active scheduler asks the OS to be put to sleep until something happens.
- 2 When it wakes up again, it knows something has happened. This something will have modified the `iStatus` field of one or more of the registered `CActive` objects. The `CActiveScheduler` simply scans its list of active objects until it finds one whose `iStatus`-field is **not** `KRequestPending`.
- 3 That objects `iActive` field is then cleared, and its `RunL` method is called.
- 4 Go back to 1 above and repeat.

(If, in step 2, the `CActiveScheduler` can find no objects with changed `iStatus` and panics the thread.)

Thank you.