Symbian OS C++ for Mobile Phones Volume 2

Programming with extended functionality and advanced features

Richard Harrison

With

Alan Robinson, Arwel Hughes, Dominic Pinkman, Elisabeth Måwe, Gregory Zaoui, Nick Johnson, Richard Potter

Reviewed by

Alex Peckover, Alex Wilbur, Chris Trick, Dan Handley, John Roe, Leon Bovett, Murray Read, Nick Tait, Paul Hateley

Managing editor

Phil Northam

Assistant editor

Freddie Gjertsen

John Wiley & Sons, Ltd

Symbian OS C++ for Mobile Phones Volume 2

TITLES PUBLISHED BY SYMBIAN PRESS

- Wireless Java for Symbian Devices
Jonathan Allin
0471 486841 512pp 2001 Paperback

- Symbian OS Communications Programming
Michael J Jipping
0470 844302 418pp 2002 Paperback

- Programming for the Series 60 Platform and Symbian OS
Digia
0470 849487 550pp 2002 Paperback

- Symbian OS C++ for Mobile Phones, Volume 1
Richard Harrison
0470 856114 826pp 2003 Paperback

- Programming Java 2 Micro Edition on Symbian OS
Martin de Jode
0470 092238 498pp 2004 Paperback

- Symbian OS C++ for Mobile Phones, Volume 2
Richard Harrison
0470 871083 448pp 2004 Paperback

- Symbian OS Explained
Jo Stichbury
0470 021306 448pp 2004 Paperback

Symbian OS C++ for Mobile Phones Volume 2

Programming with extended functionality and advanced features

Richard Harrison

With

Alan Robinson, Arwel Hughes, Dominic Pinkman, Elisabeth Måwe, Gregory Zaoui, Nick Johnson, Richard Potter

Reviewed by

Alex Peckover, Alex Wilbur, Chris Trick, Dan Handley, John Roe, Leon Bovett, Murray Read, Nick Tait, Paul Hateley

Managing editor

Phil Northam

Assistant editor

Freddie Gjertsen

John Wiley & Sons, Ltd

Published by John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester,
West Sussex PO19 8SQ, England
Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.co.uk
Visit our Home Page on www.wileyeurope.com or www.wiley.com

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 33 Park Road, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario,
Canada M9W 1L1

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data

Harrison, Richard.
Symbian OS C++ for mobile phones / By Richard Harrison.
p. cm.
Includes bibliographical references and index.
ISBN 0-470-85611-4 (Paper : alk. paper)
1. Cellular telephone systems – Computer programs. 2. Operating systems (Computers) I. Title.
TK6570.M6H295 2003
621.3845'6 – dc21

03006223

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0-470-87108-3

Typeset in 10/12pt Optima by Laserwords Private Limited, Chennai, India
Printed and bound in Great Britain by Biddles Ltd, King's Lynn
This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

Contents

Foreword

David Wood, Executive Vice President, Research, Symbian

Less than eighteen months have passed since the appearance of the first volume of *Symbian OS C++ for Mobile Phones*. These eighteen months have seen giant strides of progress for the Symbian ecosystem. In 2003 alone, the number of commercially available add-on applications for Symbian OS phones tripled. In the fourth quarter of that year, an unprecedented number of distinct new Symbian OS phone models – eight – reached the market. And in December of that year, for the first time, over a million phones running Symbian OS shipped in a single month. Looking ahead, five different 3G Symbian OS phones have recently reached shops around the world, underscoring Symbian's leadership position for the emerging generation of mobile phones. New licensing deals have been announced with premier companies in Japan, China, Korea and Taiwan, highlighting the global interest in the capabilities of the Symbian ecosystem. Last but not least, the Symbian Enterprise Advisory Council has been formed, in which leading providers of mobile business solutions are actively collaborating to promote the rapid take-up of Symbian OS phones for business use.

The good news for Symbian OS developers is that, despite these dramatic changes, the basics of the Symbian development world remain the same. Applications written to run on Symbian OS phones in 2003 will

also run on Symbian OS phones reaching the market in 2004 and 2005, in most cases with very few changes and optimizations (and in many cases with no changes required at all). Symbian OS is written in a style of C++ that holds consistently throughout all levels of the software, and throughout all versions of the operating system. Once you learn the rules, you find they apply far and wide. Symbian OS was deliberately designed to be future-proof – to 'expect the unexpected'. As the first waves of the 3G future reach us, it is reassuring to see how well the programming framework thrives despite all the changes.

Over the last eighteen months, Symbian's 500-strong team of in-house software engineers has considerably extended the scope and functionality of Symbian OS. Volume 2 of *Symbian OS C++ for Mobile Phones* is your chance to boost your own understanding of the resulting prodigious software suite. This book builds on the foundations of its predecessor, covering some of the pivotal features of Symbian OS in more detail, and goes on to describe the key new software features which are now appearing in the latest breakthrough phones.

Symbian provides the platform that enables innovation through openness; developers such as the readers of this book provide the ingenuity and the diverse domain knowledge to create myriad solutions. It is my fervent wish that software which you write, with this book as your guide, becomes dear to millions of users of Symbian OS phones.

About This Book

Symbian OS C++ for Mobile Phones Volume 2 provides information in three main areas:

1. It provides a comprehensive review of the basic techniques needed to program a Symbian OS application. The descriptions are supplemented by many straightforward and easy-to-follow examples, which range from code snippets to full applications.
2. It promotes further understanding of Symbian OS, by describing the interaction between an application and the underlying system code. This theme pervades the whole book, but a particular example is Chapter 3, which provides an illuminating walk-through of the lifecycle of a typical application, from startup to closedown.
3. It describes some of the significant new features that are introduced in Symbian OS v7.0s. This aspect is particularly significant in the discussion of multimedia services in Chapter 7, and in Chapter 8, which provides an up-to-date description of the use of Symbian OS communications and messaging services.

Symbian OS is used in a variety of phones with widely differing screen sizes. Some have full alphanumeric keyboards, some have touch-sensitive screens and some have neither. As far as possible, the material in this book is independent of any particular user interface. However, real applications run on real phones so, where necessary, we have chosen to use the Series 60 user interface and the Nokia 6600 phone as concrete examples. Wherever relevant, the text explains the principal differences between the Series 60 and UIQ user interfaces. This kind of information is invaluable for anyone who wishes to create versions of an application to run on a variety of Symbian OS phones.

Symbian OS C++ for Mobile Phones Volume 2 complements the Symbian OS software development kits. When you've put this book

down, the exclusive Symbian OS v7.0s TechView SDK supplied will be your first resource for reference information on the central Symbian OS APIs. For more specialized and up-to-date information relating to a specific mobile phone, you will need to refer to a product-specific SDK, available from the relevant manufacturer.

These SDKs contain valuable guide material, examples and source code, which together add up to an essential developer resource. As a general rule, if you have a query, look first at the SDK: you'll usually find the additional information you need that takes things further than we could in just one book.

Conventions

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.

These boxes hold important, not-to-be-forgotten information that is directly relevant to the surrounding text.

While this style is used for asides to the current discussion.

We use several different fonts in the text of this book:

- When we refer to words you use in your code, such as variables, classes and functions, or refer to the name of a file, we use this style: `iEikonEnv`, `ConstructL()`, or `e32base.h`.
- URLs are written like this: ***www.symbiandevnet.com***.
- And when we list code, or the contents of files, we'll use the following convention:

```
Lines that show concepts directly related to the surrounding text are
shown on a gray background
```

```
But lines which do not introduce anything new, or which we have seen
before, are shown on a white background.
```

- We show commands typed at the command line like this:

```
abld build winscw udeb
```

Innovation Through Openness

The success of an open operating system for smartphones is closely linked to the degree to which the functionality of lower levels of software and hardware can be accessed, modified, and augmented by add-on software and hardware. As Symbian OS smartphones ship in volume, we are witnessing the arrival of a third wave of mobile phones.

The first wave was voice-centric mobile phones. Mobile phone manufacturers have performed wonders of optimization on the core feature of these phones – their ability to provide great voice communications. Successive generations of products improved their portability, battery life, reliability, signal handling, voice quality, ergonomics, price, and usability. In the process, mobile phones became the most successful consumer electronics product in history.

The second wave was rich-experience mobile phones. Instead of just conveying voice conversations between mouth and ear, these phones provided a much richer sensory experience than their predecessors. High-resolution color screens conveyed data vividly and graphically. High-fidelity audio systems played quality music through such things as ringtones and audio files. These phones combined multimedia with information and communications, to dramatic effect.

But the best was still to come. The primary characteristic of the third wave of mobile phones is their openness. Openness is an abstract concept, but one with huge and tangible consequences for developers. The key driver is that the growing on-board intelligence in modern phones – the smartness of the hardware and software – can now be readily accessed by add-on hardware and software. The range of applications and services that can be used on a phone is not fixed at the time of manufacture, meaning new applications and services can be added afterwards. The phone can be tailored by an operator to suit its customers and these customers can then add further customizations, reflecting specific needs or interests.

The Symbian Ecosystem

Open phones allow a much wider range of companies and individuals to contribute to the value and attractiveness of smartphones. The attractiveness of a phone to an end-user is no longer determined only by the various parties involved in the creation of that phone. Over-the-air downloads and other late-binding mechanisms allow software engineers to try out new ideas, delivering their applications and services directly to end-users. Many of these ideas may seem unviable at time of manufacture. However, the advantage of open phones is that there is more time, and more opportunity, for all these new and innovative ideas to mature into advantageous, usable applications that can make a user's life easier – whether it be over-the-air synchronization with a PC, checking traffic or having fun with 3D games or photo editing.

The real power of open phones arises when add-on services developed for a phone are reused for add-on services on other phones. This allows an enormous third-party development ecosystem to flourish. These third parties are no longer tied to the fortunes of any one phone, or any one phone manufacturer. Moreover, applications that start their lives as add-ons for one phone can find themselves incorporated at time of manufacture in subsequent phones, and be included in phones from other manufacturers. Such opportunities depend on the commonality of the underlying operating system. Open standards drive a virtuous cycle of research and development: numerous companies that can leverage the prowess, skills, experience and success of the Symbian ecosystem.

Symbian OS Phones

This book focuses on Symbian OS v7.0s, and the additional technologies it brings to mobile phones, as well as expanding on the core programming techniques explored in *Symbian OS C++ for Mobile Phones Volume 1*. We use the first Symbian OS v7.0s phone, the Nokia 6600, to illustrate most of the examples in the text, but these should also be demonstrable in Symbian OS v7.0 as well as other Version 7.0s phones. Phones following on from the Nokia 6600 include the Nokia 7700, 7610 and 9500, Panasonic X700 and Samsung SGH-D710.

Symbian OS phones are currently based on the following user interfaces open to C++ and Java programmers: the Series 80 Platform (Nokia 9200/9500 Communicator series), the Series 90 Platform (Nokia 7700), the Series 60 Platform (Nokia 7610, 6600, 6620, 7650, 3650, 3660, 3620, N-Gage, Siemens SX1, Sendo X, Panasonic X700 and Samsung SGH-D710), and UIQ (Sony Ericsson P800, P900, BenQ P30, Motorola A920, A925, A1000). The Nokia 6600 was the first smartphone to include

Java MIDP 2.0. Read on for a brief summary of the user interface families now available.

Mobile Phones with a Numeric Keypad

These phones are designed for one-handed use and require a flexible UI that is simple to navigate with a joystick, softkeys, jogdial, or any combination of these. Examples of this come from the Series 60 Platform. Fujitsu produces a user interface for a range of phones including the F2102v, F2051 and F900i for NTT DoCoMo's FOMA network. Pictured is the Siemens SX1.

Mobile Phones with Portrait Touch Screens

These mobile phones tend to have larger screens than those in the previous category and can dispense with a numeric keypad altogether. A larger screen is ideal for viewing content or working on the move, and pen-based interaction gives new opportunities to users and developers. The best current example of this form factor is UIQ, which is the platform for the Sony Ericsson P800 and P900, as well as BenQ P30 and Motorola's A920, A925 and A1000. The P800, P900 and P30 actually combine elements of full screen access and more traditional mobile phone use by including a numeric keypad, while the Motorola smartphones dispense with a keypad altogether. Pictured is the Sony Ericsson P900.

Mobile Phones with Landscape Screens

These mobile phones have the largest screens of all Symbian OS phones and can have a full keyboard and could also include a touch screen. With this type of mobile phone developers may find enterprise applications particularly attractive. A current example of the keyboard form factor is the

Series 80 Platform. This UI is the basis of the Nokia 9200 Communicator series, which has been used in the Nokia 9210i and Nokia 9290 and will be used in the Nokia 9500. Based on Series 90, the Nokia 7700 is an example of a touch screen mobile phone without keyboard aimed more at high multimedia usage.

When you're ready to use the Symbian OS C++ programming skills you've learned in this book, you'll want an up-to-the-minute overview of available phones, user interfaces and tools. For the latest information, start at ***www.symbian.com/developer*** for pointers to partner websites, other books, white papers and sample code. If you're developing technology that could be used on any Symbian OS phone, you can find more information about partnering with Symbian at ***www.symbian.com/partners***.

We wish you an enjoyable experience programming with Symbian OS and lots of success.

About the Authors

Richard Harrison, Lead Author

Richard joined Symbian (then known as Psion) in 1983 after several years teaching maths, physics and computer science. During that time he wrote a Forth language implementation for Acorn Computers, and wrote accompanying user manuals for the Acorn Atom and BBC Micro.

He has spent the majority of his time in system integration (SI), building and leading the SI team. He has produced user documentation for software for the Sinclair QL, the PC application software for the Psion Organiser I and the source code translator for the original version of OPL. Joint author of the Organiser II spreadsheet and principal designer and author of the Psion Series 3 and 3a word processors, he was also lead author of the Psion SIBO SDK team.

Educated at Balliol College, Oxford, with an MA in Natural Science (Physics), Richard also graduated from Sussex University with an MSc in Astronomy, and spent a further two years of postgraduate research in the Astronomy Group at Imperial College.

Alan Robinson

Alan Robinson joined Symbian shortly after its formation in 1998 and has mostly worked on documentation and examples in messaging and communications. Alan previously contributed to *Wireless Java for Symbian Devices* (Wiley, 2001) and *Symbian OS C++ for Mobile Phones Vol 1* (Wiley, 2003).

A graduate of Cambridge University with a BA in literature and philosophy, he became interested in applying logical theory and took a Computing MSc at Middlesex University. He has worked on developer kits for a startup company's messaging middleware platform, and for IBM's MQ Series.

Arwel Hughes

Arwel joined Symbian (then Psion) in 1993, working on documentation for the Series 3a and also some software development. Since the formation of Symbian, he has contributed documentation and examples for Symbian OS. This is rather like painting the famous Forth Bridge: just when you think you can see the end...

Arwel previously worked on IBM mainframes in roles including programmer and systems programmer for a number of companies including GKN, Prudential Assurance, Shell and Chase Manhattan Bank. He has a BSc in Applied Mathematics from Sheffield University.

Dominic Pinkman

Dominic joined Psion in October 1995 as a technical author. He has written and maintained documentation for APIs throughout Symbian OS, and was a co-author of the book *Symbian OS C++ for Mobile Phones Vol 1* (Wiley, 2003).

He has an MSc in Computer Science from the University of Kent and a BA in Modern Language studies from Leicester University.

Elisabeth Måwe

Elisabeth joined the system documentation team in 2000 and has since been designing and writing the Symbian Developer Library, specializing in operating system customization, kits, emulators, test, build and release tools. She has also been involved in training and usability management.

Elisabeth has a BA in Technical Communication/Information Design from Mälardalens Högskola and Coventry University, as well as an MA in Contemporary English Language and Linguistics from Reading University. After graduating in 1996 she worked as a technical author, information designer and web editor for various IT companies in the UK, producing documentation for both network management and market research software. She would like to thank Alex Peckover and Murray Read for providing both example code and technical expertise.

Greg Zaoui

Gregory Zaoui first joined Symbian in 1998, as a graduate software engineer with a 'Licence de Mathématiques' from the University of Strasbourg. He has been working on various projects for System Integration on build tools and release management. He then joined the newly created Test Solutions group in 2002, as a technical architect for TechView and other test tools.

His interests range from skiing and windsurfing to talmudic studies. Gregory would like to thank Richard Harrison and Paul Treacy for their

excellent mentoring, as well as Clare Oakley (Test Solutions manager) without whom it would be impossible to talk consistently about test tools for Symbian OS. He also would like to acknowledge Elisabeth Måwe for her very active participation to the chapter, Konstantin Michaelov for his very useful example cases, Andrew Thoelke for the profiler bits, and all Test Solutions developers for their contribution. Gregory would also like to add special thanks to his dear wife Tamar for her constant encouragement and most precious help.

Nick Johnson

Since joining Symbian, Nick worked for a year in the Multimedia team helping implement next-generation Multimedia APIs and frameworks on Symbian OS and then subsequently transferred to Symbian's Marketing department, where he is now working as a developer consultant assisting Symbian partners with their Multimedia troubles.

Previous to this, Nick first spent three years studying Computer Science and Cybernetics at Reading University before subsequently spending two years working in 3D sound research at Sensaura Ltd. Here he spent time both developing new 3D sound algorithms and implementing the Xbox and GameCube ports of their cross-platform 3D audio middleware library. After leaving Sensaura, Nick spent a few weeks in the games industry working on 'Microsoft Train Simulator 2' before deciding that it just wasn't for him and instead joined Symbian.

Outside work, Nick enjoys learning Japanese and about Japanese culture, is a home cinema/film enthusiast, enjoys collecting/drinking rare liquors and vodkas and spends large amounts of time trying to convince friends that LaserDiscs are still the way forward . . .

Richard Potter

Richard joined Symbian in the summer of 2002 as a technical author. He works on documentation for the Security and Networking subsystems and has also written some Perl and Python scripts to aid the team.

Richard's unusual route to becoming a technical author includes advertising photography, being a singer/guitarist in a rock band, an MSc in Astrophysics, and an MPhil in Experimental Particle Physics working at the Stanford Linear Accelerator Center, Palo Alto, California. Many thanks to Jelte Liebrand for his advice.

Acknowledgements

Many thanks, in no particular order, to Marit Doving, Ian Weston, Omid Rezvani, Jason Dodd, Ade Steward, Ski Club, Iain Dowie, Sander Siezen, Nick 'I, Robot' Tait, Colin Turfus, Martin de Jode, Dave Jobling, Bart Govaert, Phil 'Ooc Clavdivs' Spencer, Karen Mosman, Colin Anthony, and System Management Group for the Symbian OS system model. Their contributions and support have all been very much appreciated. Much respect to the Laughing Gravy and Dingo Dave at the Stage Door for providing vital fuel. Original cover concept by Jono Tastard.

About the Cover

The mobile phone has traditionally connected the mouth to the ear – at Symbian's Exposium 2003, Symbian introduced the concept of Symbian OS enabling a new generation of connected communications devices by connecting the mouth to the ear to the eye. To realize this vision, the mobile phone industry is working together through Symbian to develop the latest technologies, support open industry standards, and ensure interoperability between advanced mobile phones as networks evolve from 2.5G to 3G and beyond . . .

Symbian licenses, develops and supports Symbian OS, the platform for next-generation data-enabled mobile phones. Symbian is headquartered in London, with offices worldwide. For more information see the Symbian website, ***http://www.symbian.com/***. 'Symbian', 'Symbian OS' and other associated Symbian marks are all trademarks of Symbian Software Ltd. Symbian acknowledges the trademark rights of all third parties referred to in this material.

1

Symbian OS Fundamentals

Before we head into the deeper aspects of Symbian OS, we need to spend some time looking at some of the basic operations, programming patterns and classes that are common to all applications, and indeed, to almost all code that runs in the system.

What we are going to see here are the basic patterns that are used over and over again: building blocks that allow you to build safe and efficient code.

Symbian OS uses object-orientation, and is written in C++, with a tiny bit of assembler thrown in at the lowest levels. This means that the vast majority of applications are written in C++.

The use of C++ in Symbian OS is not exactly the same as C++ in other environments:

- C++ does more than Symbian OS requires – for example, full-blown multiple inheritance.
- C++ does less than Symbian OS requires – for example, it doesn't insist on the number of bits used to represent the basic types, and it knows nothing about DLLs.
- Different C++ communities do things differently because their requirements are different. In Symbian OS, large-scale system design is combined with focus on error handling and cleanup, and efficiency in terms of ROM and RAM budgets.

1.1 Object Creation and Destruction

One of the fundamental characteristics about object-oriented systems is the creation and destruction of objects. Objects are created, have a finite lifetime, and are then destroyed.

Symbian OS C++ for Mobile Phones, Volume 2. Edited by Richard Harrison
 ISBN: 0-470-87108-3

Object creation and destruction is intimately tied up with the issue of cleanup, making sure that your applications are coded in such a way that they do not leak memory – a real issue for systems that may not be rebooted for long periods, if at all.

We'll first just look at the very basics of object creation and destruction in Symbian OS. In some ways this may give a misleading picture – the full picture will only emerge once we've looked at error handling and cleanup. This is because object creation, object destruction, error handling and cleanup are all intimately tied together with the aim of ensuring that objects, once created, are always destroyed when no longer needed.

There are two places in Symbian OS where you can create objects: the heap and the program stack.

1.1.1 The Heap (Dynamic Objects)

All threads have an associated heap, termed the default heap, from which memory can be allocated at runtime. This is where you put large objects, and objects that can only be built at runtime, including dynamic variable length strings. This is also where you put objects whose lifetimes don't coincide with the function that creates them – typically such objects become data members of the parent or owning object, with the relationship expressed as a pointer from owning object to owned object.

Memory is allocated from the thread's default heap, as and when required, using the C++ operator `new` and, very rarely, using user library functions such as `User::Alloc()`. If there is insufficient free memory, then an allocation attempt fails with an out-of-memory error.

In Symbian OS, classes that are intended to be instantiated on the heap are nearly always derived from the `CBase` class. This class gives you two things:

- zero initialization, so that all data members, member pointers and handles are initially zero
- a virtual destructor, so that the object can be properly destroyed. This is an important point when we come to look at cleanup issues later.

Strictly speaking, `CBase` is a base class for all classes that own resources, for example other bits of heap, server sessions, etc. What this means is that all `CBase` derived classes must be placed on the heap, but that not all heap objects are necessarily `CBase` derived.

The following code fragment shows a simple way of creating a heap-based object:

```
class CMyClass : public CBase
    {
public:
    CMyClass();
    ~CMyClass();
    void Foo();
private:
    TInt     iNumber;
    TBuf<32> iBuffer;
    }
```

```
CMyClass* myPtr = new CMyClass;
if (myPtr)
     {
     myPtr->Foo();  // can safely access member data & functions
     }
delete myPtr;
```

If there is insufficient memory to allocate the `CMyClass` object, then `myptr` is NULL. If allocation succeeds, `myPtr` points to the new `CMyClass` object, and further, the data members `iNumber` and `iBuffer` are *guaranteed* to be binary zeroes. Conversely, the `delete` operator causes the object's destructor to be called before the memory for the object itself is released back to the heap.

There's one very important variation on this. Take a look at the following code:

```
CMyClass* myPtr = new (ELeave) CMyClass;
myPtr->Foo(); // can safely access member data & functions
...
delete myPtr;
```

The main difference here is that we have specified `ELeave` as part of the new operation. What this means is that instead of returning a NULL value when there isn't enough memory in which to create the `CMyClass` object, the operation 'leaves'. We'll explore what leaving means in more detail later when we investigate error handling and cleanup, but for the moment, think of it as an operation where the function returns immediately.

If the new operation doesn't leave, then it means that memory allocation for the new object has succeeded, the object has been created, and program control flows to the next C++ instruction, that is, the instruction `myPtr->Foo()`. It also means that there's no need to check the value of `myPtr` – the fact that the new operation returns means that `myPtr` will have a sensible value.

1.1.1.1 Ownership of Objects

In a typical object-oriented system such as Symbian OS, where objects are created dynamically, the concept of ownership is important. All objects need to be unambiguously owned so that it is clear who has responsibility for destroying them.

Use a destructor to destroy objects that you own.

1.1.1.2 Don't Forget About Objects – Even by Accident

Don't allocate objects twice. It sounds obvious, but allocating an object a second time, and putting the address into the same pointer variable into which you put the address of the first allocated object, means that you lose all knowledge of that first object. There is no way that a class destructor – or any other part of the C++ system – can find this object, and it represents a memory leak.

1.1.1.3 Deleting Objects

As we've seen, deleting an object is simply a matter of using the `delete` operator on a pointer to the object to be deleted. If a pointer is already zero, then calling `delete` on it is harmless. However, you must be aware that `delete` does not set the pointer itself to zero. While this does not matter if you are deleting an object from within a destructor, it is very important if the deletion occurs anywhere else. Double deletion doesn't always cause an immediate crash, and sometimes it leaves side-effects that only surface a long time after the real problem – the double delete – occurred. As a result, double deletes are very hard to debug.

On the other hand, double deletes are easy to avoid – just follow this little discipline:

> **C++ `delete` does not set the pointer to zero. If you delete any member object from outside its class's destructor, you must set the member pointer to NULL.**

1.1.2 The Program Stack (Automatic Objects)

The stack is used to hold the C++ automatic variables for each function. It is suitable for fixed size objects whose lifetimes coincide with the function that creates them. In Symbian OS, the stack is a limited resource. A thread's stack cannot grow after a thread has been launched; the thread is panicked – terminated abruptly – if it overflows its stack. This means that stack objects in Symbian OS shouldn't be too big, and they should

only be used for *small* data items – for example, strings of a few tens of characters, say. Taking string data as an example, a good rule of thumb is to put anything larger than a file name on to the heap. However, it's quite acceptable to put pointers (and references) onto the stack – even pointers to very large objects.

You can control the stacksize in a `.exe`, through the use of the `epocstacksize` keyword of the `.mmp` file used to create the `.exe`. However, this only applies to console programs, servers or programs with no GUI – and not to GUI programs as they are launched with `apprun.exe`. GUI programs have a small program stack, of the order of 20k, and must be considered a valuable resource.

You can control the stacksize when you launch a thread explicitly from your program. However, avoid the temptation to create a large stack as this will eat into valuable resources.

We put built-in types, or classes that don't need a destructor, on to the program stack. They don't need a destructor because they own no data. This means that they can be safely discarded, without the need for any kind of cleanup. You simply exit from the function in which the automatic variable was declared. The type of objects that can go on to the stack are:

- any built-in type: these are given `typedefs`, such as `TInt` for a signed integer.
- any enumerated type, such as `TAmPm`, which indicates whether a formatted time-of-day is am or pm. Note that all enumeration types have names starting with a `T`, though enumerated constants such as `EAm` or `EPm` begin with `E`.
- class types that do not need a destructor, such as `TBuf<40>` (a buffer for a maximum of 40 characters) or `TPtrC` (a pointer to data, or to a string of any number of characters). `TPtrC` contains a pointer, but it only *uses* (rather than *has*) the characters it points to, and so it does not need a destructor.

For example, given a function `Foo()` in class `CMyClass`, we can create a `TInt` and a `TBufC<16>` type as automatic variables, use them in the body of the function, and then simply discard them, without doing any kind of cleanup when the function exits.

```
void CMyClass::Foo()
    {
    TInt myInteger;
    TBufC<16> buffer;
    ...
    // main body of the function
    } // variables are 'lost' on exit from the function.
```

1.2 Error Handling and Cleanup

In machines with limited memory and resources, such as those that Symbian OS is designed for, error handling is of fundamental importance. Errors are going to happen, and you can't afford not to handle them correctly.

Symbian OS provides a framework for error handling and cleanup and is a vital part of the system with which you need to become familiar. Every line of code that you write – or read – will be influenced by thinking about cleanup. No other Symbian OS framework has so much impact; cleanup is a fundamental aspect of Symbian OS programming. Because of this, we've made sure that error handling and cleanup are effective and very easy to do.

1.2.1 What Kinds of Error?

The easiest way to approach this is by focusing on out-of-memory errors.

These days, desktop PCs come with at least 256 MB of RAM, virtual memory swapping out on to 20 GB or more of hard disk, and users who expect to perform frequent reboots. In this environment, running out of memory is rare, so you can be quite cavalier about memory and resource management. You try *fairly* hard to release all the resources you can, but if you forget then it doesn't matter too much: things will get cleaned up when you close the application, or when you reboot. That's life in the desktop world.

By contrast, Symbian OS phones have as little as 4 MB of RAM, and often no more than 16 MB, although there are now devices with 32 MB. Nevertheless, by comparison with a PC, this is small; there is no disk-backed virtual memory. Remember that your users are *not* used to having to reboot frequently.

You have to face some key issues here – issues that don't trouble modern desktop software developers:

- You have to program efficiently, so that your programs don't use RAM unnecessarily.
- You have to release resources as soon as possible, because you can't afford to have a running program gobble up more and more RAM without ever releasing it.
- You have to cope with out-of-memory errors. In fact, you have to cope with potential out-of-memory for *every single operation* that can allocate memory, because an out-of-memory condition can arise in any such operation.