Programming PC Connectivity Applications for Symbian OS

Smartphone Synchronization and Connectivity for Enterprise and Application Developers

By

Ian McDowall

Reviewed by

Day Barr, Emlyn Howell, Helena Bryant, Paul Newby, Rob Falla, Simon Didcote, Tony Naggs, Zoë Martin

Symbian Press

Managing editor

Phil Northam

Project editor

Freddie Gjertsen

John Wiley & Sons, Ltd

Programming PC Connectivity Applications for Symbian OS

TITLES PUBLISHED BY SYMBIAN PRESS

- Programming PC Connectivity Applications for Symbian OS
Ian McDowall
0470 090537 477pp 2004 Paperback

- Symbian OS Explained
Jo Stichbury
0470 021306 416pp 2004 Paperback

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

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

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

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

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

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

Programming PC Connectivity Applications for Symbian OS

Smartphone Synchronization and Connectivity for Enterprise and Application Developers

By

Ian McDowall

Reviewed by

Day Barr, Emlyn Howell, Helena Bryant, Paul Newby, Rob Falla, Simon Didcote, Tony Naggs, Zoë Martin

Symbian Press

Managing editor

Phil Northam

Project editor

Freddie Gjertsen

John Wiley & Sons, Ltd

 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

McDowall, Ian.
Programming PC connectivity applications for Symbian OS : smartphone synchronization and connectivity for enterprise and application developers / by Ian McDowall.
p. cm.
Includes bibliographical references and index.
ISBN 0-470-09053-7 (pbk. : alk. paper)
1. Cellular telephone systems – Computer programs. 2. Operating systems (Computers) 3. Computer input-output equipment. I. Title.
TK6570.M6M38 2004
005.26'8 – dc22

2004017257

British Library Cataloguing in Publication Data

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

ISBN 0-470-09053-7

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

Author Biography

Ian joined Symbian in 2000 and is currently a technology architect responsible for connectivity. He has previously filled roles ranging from developer through project manager to technical manager by way of quality manager and process consultant (including presentation at international conferences).

He has an MA in Computer Sciences from Cambridge University and an MBA from Warwick University. As a software engineer for over twenty years he has been with a number of software companies and has worked on more than fifteen operating systems, developing software ranging from enterprise systems to embedded software. He is married to Lorraine and they have two children, Ross and Kelly, and a number of pets.

Author's Acknowledgments

I would like to thank the members of the PC Connectivity team and others in Symbian's Software Engineering Department who have made this book possible. In the PC Connectivity team Day Barr, Simon Didcote and Paul Newby have provided essential information and suggestions, and in other teams Emlyn Howell, Tony Naggs and David Cunardo have provided invaluable advice on the best use of Symbian's Messaging, Contacts and Agenda APIs.

I would like to thank Zoë Martin, Colin Turfus and Ian Weston for their support in promoting the wider use of PC Connectivity software.

The other reviewers have also been both diligent and constructive – Helena Bryant and Rob Falla (who also suggested the original idea for this book).

I must thank Freddie Gjertsen and Phil Northam of Symbian Press for promoting the concept of this book inside and outside Symbian and for their patient checking and support.

I would also like to thank all the engineers in Symbian and elsewhere who have made Symbian's PC Connectivity software what it is today.

Finally, I would like to thank my family who have put up with my work on this book for more than a year.

Symbian Press Acknowledgments

Symbian Press would like to thank Ian for his perseverance in adversity. And all those who reviewed the book, mentioned or otherwise. And those who worked 'behind the scenes' to allow this book to be realized. And, of course, the BA cabin-crew for always looking after the Symbian Press 'frequent flyer' so splendidly. And the wonderful Loza, Symbian Press Officer extraordinaire.

Cover concept by Jonathan Tastard.

1

Introduction

Welcome to this book on programming PC Connectivity applications for Symbian OS. PC Connectivity applications based on standard services and APIs can be created purely by programming on the PC, but more specialized applications involve programming on the Symbian OS smartphone as well as on the PC. This book will help you to create both types of application.

If you have created an application for Symbian OS, have you considered how to improve its usability by integrating it with a PC? Maybe your application could support a user interface on the PC when the Symbian OS smartphone is connected, or maybe your application could store or archive data on the PC.

If you have created an application for Windows PCs, have you considered how to improve your application by integrating it with Symbian OS smartphones? This has been considered difficult and expensive but, with the information in this book, it can be straightforward. You may be surprised at the quality of integration you can achieve just by creating PC software – for example, you could manage media files such as image, audio and video files just by using the APIs described in Chapters 4 and 5. If your application is more specialized then a small amount of Symbian OS programming may give you a unique service that increases its attractiveness (and therefore its sales).

All the examples in this book are of stand-alone PC Connectivity applications, but this is by no means the only way to create PC Connectivity applications. We will create a file browser that will provide a convenient user interface to the filing system on Symbian OS smartphones; we will create an application to read SMS messages on the smartphone and to send such messages by means of the smartphone; we will create applications to directly read and modify the Contacts and Agenda data on the smartphone.

These applications are potentially useful (I certainly use them extensively in favor of the other ways of accessing Contacts, Agenda and SMS

messages) but they are just examples of what can be done and I have deliberately kept them simple. If you want to create a fully-featured, integrated and commercialized version of these applications then I wish you good luck. However, I feel that the largest potential value of PC Connectivity applications lies in integration with other applications.

If you are an Enterprise or corporate developer this book is also aimed at you. Symbian OS provides a selection of methods to connect a smartphone to a server of which PC Connectivity is one of the cheapest and fastest.

1.1 What is PC Connectivity and Why is This Book Different from Other Symbian OS Books ?

A PC Connectivity application is an application with one part on the Symbian OS smartphone and one part on the PC. Usually, the software on the smartphone (commonly referred to as a server or service) will not have a user interface.

Most books on Symbian OS programming are concerned with developing applications with user interfaces (although some also cover server design). This book contains no information on user interface programming on Symbian OS, but it does provide specialist information on how to create services using the PC Connectivity plug-in and server APIs. As most PC Connectivity services are interfaces to existing servers on the smartphone, this book does not go into detail about server design.

The most obvious PC Connectivity software that most users will see is the PC suite that is supplied with their Symbian OS smartphone. Typical functions supported by such a PC suite include:

- synchronization with a Personal Information Manager (PIM) such as Microsoft Outlook or Lotus Notes
- backup and restore of data
- the ability to install software on the smartphone remotely from the PC.

Some manufacturers add extra features such as a configuration application or an interface to an MP3 player.

Some other books on Symbian OS programming touch on PC Connectivity, but they only describe the standard functions and do not provide information on how to create new PC Connectivity applications.

This book is aimed at software engineers creating additional applications or integrating other applications; it is not aimed at smartphone manufacturers (who have their own support channels). This book does not, therefore, cover the standard functions listed above, as I do not expect third-party developers to create alternative PC suites.

1.2 What This Book Will Tell You (and What It Will Not)

This book covers PC Connectivity for a wide range of Symbian OS smartphones but, unfortunately, not all. The reasons for the variations in PC Connectivity framework are discussed in Chapter 2. In summary, this book covers all Symbian OS smartphones that I know of based on Symbian OS v6.1 and Symbian OS v7.0 and some smartphones based on Symbian OS v7.0s. It will also apply to a large extent to smartphones based on Symbian OS v8.0 and later that include Symbian's TCP/IP PC Connectivity framework.

Because this book is aimed at PC Connectivity developers, it has less space for general-purpose Symbian OS programming than some other (commonly much larger) books. Chapter 6 explains the basics of Symbian OS programming and the later chapters cover PC Connectivity and some other APIs in detail. The developer resources that are available will provide more detail on the other APIs in Symbian OS. However, if you want to go further into Symbian OS programming then you may benefit from another book that concentrates on Symbian OS programming, such as other books published by Symbian Press.

For the Symbian OS programming I have used C++. This is the language used to develop Symbian OS and the language in which all the APIs are provided. Symbian uses C++ in ways that are slightly different from how the language is used with other operating systems, but a good grasp of object-oriented programming will be essential for any Symbian OS development.

It is certainly possible to program for Symbian OS using Java. Java is well supported by Symbian OS and is the most appropriate language for a range of purposes, but there is no PC Connectivity API for Java. However, Chapter 14 does show a way in which Java services might be accessed using a PC Connectivity framework.

For programming on the PC I have used C# as I think it is the best current language for development in Microsoft Windows. However, it is possible to use any Microsoft-supported development language as the PC Connectivity framework uses COM. The examples in this book would be easily understood by any Microsoft Visual C++ programmer and, I believe, also by Visual Basic programmers.

Because space in this book is limited, I have not provided any tuition in using C++ or C#. Many good books and online resources on both languages are available.

I have not included extensive information on using some of the programming tools such as Integrated Development Environments (in particular the debuggers). This is because I assume that the reader will already know how to use these tools.

This book is aimed at integrating Symbian OS smartphones with PCs running Microsoft Windows (multiple versions). What about other

operating systems such as Linux or Mac OS? Theoretically, there is no reason why PC Connectivity applications could not be created on these operating systems, but Symbian does not publish the protocols required and so the possibility remains more theoretical than practical. I have created some experimental PC Connectivity software on Linux using Perl, but that used undocumented protocols, required manual configuration and would not work on all bearers.

1.3 How This Book is Structured

This book is structured so as to be read from the start to the end, but it can also be used as a reference book and be dipped into.

Chapter 2 describes the history of Symbian OS in general and of Symbian OS PC Connectivity in particular. You do not need to read this to use the rest of the book, but it is only a short chapter and does provide some context if you want to understand how Symbian OS evolved to its current state.

Chapter 3 describes the architecture of Symbian OS PC Connectivity and is a basis for understanding later chapters.

Chapters 4 and 5 describe the APIs provided on the PC for Symbian OS PC Connectivity and show how to use these in creating a file browser without writing any new code on the Symbian OS smartphone.

Chapter 6 leads into the development of software on the Symbian OS smartphone. It contains a compressed tutorial on developing for Symbian C++ but it ignores aspects that will not be used in developing for PC Connectivity.

Chapters 7 and 8 describe how to create PC Connectivity services using specialized APIs. Chapter 7 covers custom servers which are used in Symbian OS v6.1 to Symbian OS v7.0s, while Chapter 8 covers the socket server APIs introduced in Symbian OS v8.0. In order to illustrate the APIs we will see how to create very simple PC Connectivity plug-ins.

Chapters 9, 10, 11 and 12 cover specific APIs and show how to create PC Connectivity services using them. These chapters cover SMS messaging, the Contacts Model and the Agenda Model and show how to create PC Connectivity plug-ins to expose these APIs to the PC.

Chapter 13 builds on the previous chapters and shows how to create an application on the PC that communicates with the services created in the previous chapters to present a GUI on the PC that integrates with the Symbian OS smartphone.

Chapter 14 is a slight diversion that discusses how to manage services that were not originally design for PC Connectivity with minimal changes.

Chapter 15 finishes the book with a selection of advice on designing and developing PC Connectivity applications based on Symbian's experience over a number of years.

1.4 Conventions Used in This Book

This book has very little in the way of conventions that are not obvious. Example code is presented in a fixed-width font and is normally highlighted:

```
void CSomeClass::someMethod()
  {
  someOtherMethod();
  }
```

The same convention is used for Symbian OS C++ as for C# or C++ for the PC. It should always be clear which is meant by the context.

In order to avoid pages of uninteresting code listings, this book shows only the example code that I believe is relevant to the text. Because Symbian OS is less commonly understood and because the code is more compact, this book normally shows more complete listings of Symbian OS C++. In a few cases I show an early version of a method and then return to it later to add more code. In these cases the unchanged code is not highlighted so that the new code can be clearly seen.

Where this book describes C# GUI code I have omitted all of the code that is created by wizards.

Where classes or members are referred to in the text of the book, they are normally shown in a `fixed-width font` to highlight them.

1.5 Developer Resources

In order to develop for Symbian OS in general and for PC Connectivity in particular, you will need compilers and other development resources.

For PC development you will need standard Microsoft development tools and developer resources: see ***www.msdn.microsoft.com***.

For Symbian OS development the tools and resources are more specialized and are not all provided directly by Symbian. The best starting point for resources and partners is ***www.symbian.com/developer*** which has links to partner websites and also those resources that Symbian does not provide.

Previously Symbian has not released software development kits (SDKs) directly to developers. Instead the smartphone manufacturers have created software development kits for their phones and released those. With the creation of platforms that span multiple phones there are now software development kits for those platforms. In a new departure, the CD that accompanies this book includes an SDK for Symbian OS v7.0s based on the TechView test user interface.

At the time of writing, Nokia provides a range of developer resources via its Nokia forum ***www.forum_nokia.com*** and ***www.series60.com***, Sony Ericsson provides SDKs and Symbian OS documentation and tools via ***http://developer.sonyericsson.com***, and Sendo provides information via ***www.sendo.com/dev***.

Another site that is worth a look is ***www.newlc.com*** which is an independent developer site. This has links to partners as well as tutorials and other resources.

The link between the Microsoft development tools and the tools for development on the Symbian OS smartphones is the PC Connectivity framework on the PC. This requires an SDK that is available directly from Symbian at ***www.symbian.com/developer/downloads/tools.html***. As this SDK is quite small we have not included it on the CD that accompanies this book; you can pick up the latest version in a few minutes from the Symbian website.

2

A History of Symbian OS and PC Connectivity

2.1 A History of Symbian OS

Symbian was formed in 1998 as an unprecedented joint venture between the largest players in the mobile telephone industry. From its inception, Symbian has been dedicated to making Symbian OS available to any smartphone manufacturer – it is not restricted to the original investors. Since 1998 the number of licensees, that is smartphone manufacturers who have licensed Symbian OS, has grown. The number of smartphone models based on Symbian OS has grown at an increasing rate and so have the numbers of actual Symbian OS smartphones shipped.

Over the last few years smartphones have become more advanced, and the spread of advanced messaging and multimedia features requires more advanced operating systems than the earlier, native, mobile phone operating systems, while increasing computing power has made the hardware required more affordable.

Symbian OS is not a complete and fully defined (and therefore limited) system – instead it allows smartphone manufacturers to develop user interfaces according to their own views and allows them to add extra hardware features such as cameras and FM radios. Current user interfaces include screens of differing sizes but all in color and include a variety of input devices such as touchscreens, jog dials, softkeys, joysticks and built-in or attachable keyboards. Symbian does not take a view on which user interface is best but allows smartphone manufacturers to innovate and compete.

One possible source of confusion is the name EPOC. This is the original name for Symbian OS when it was created by Psion. ER5 stands for EPOC Release 5 which was the first real version of Symbian OS. The name EPOC still appears in some documentation, either as an anachronism or as a reference to the underlying kernel.

Symbian OS is not static but has developed through multiple versions from ER5, Symbian OS v6.0, Symbian OS v6.1, Symbian OS v7.0,

Symbian OS v7.0s and Symbian OS v8.0, with further versions under development. Each version builds on the previous versions and as much consistency as possible is maintained, allowing for the new features in each version.

The variety of possibilities that come with Symbian OS can be bewildering and manufacturers have developed platforms that include a user interface, a selection of applications and hardware interface layers that can be used to develop multiple models of smartphones. This reduces the development cost and time for new smartphones and allows developers to create applications that can be deployed on a wide range of smartphones.

2.2 PC Connectivity Using PLP

In the first versions of Symbian OS (ER5) PC Connectivity was provided by a component called PLP, which stands for Psion Link Protocol. PLP was originally designed for RS232 serial connections and was later extended to support infrared. It used proprietary software both on the PDA or smartphone and on the PC, and it used limited-sized buffers.

Using PLP, Symbian and smartphone manufacturers were able to provide the same headline functionality that is part of PC suites in later versions: access to the filing system, backup and restore, PIM synchronization and remote software install.

From the start, PLP supported plug-ins on the smartphone to make the PC Connectivity extensible. These plug-ins were called 'custom servers'. Each function required was implemented by a plug-in (some with supporting libraries). Because of the use of plug-ins, the PC Connectivity framework allowed extra features to be added and smartphone manufacturers were able to extend their PC suites. For this reason, the PC suite for the Nokia 9210 supports plug-ins within the PC software.

2.3 PC Connectivity Using TCP/IP

PLP was used in Symbian OS v6.0, but during the implementation of Symbian OS v6.1 Symbian switched to a TCP/IP based PC Connectivity framework. The switch coincided with the introduction of Bluetooth as a bearer.

Symbian did not develop the TCP/IP based bearer but instead licensed a product named m-Router from Intuwave. m-Router has components on the smartphone and on the PC. The PC components provide a PPP implementation that is lacking from Microsoft Windows.

m-Router supports TCP/IP connections that can be used for any purpose, but it also supports its own framework for loading services. In order

to make use of existing components, m-Router is able to load custom servers (by means of a plug-in named `ectcpadapter` that we will encounter again in Chapter 7).

The use of TCP/IP connections rather than PLP allowed a number of underlying technical improvements, but the same functions were supported by the PC suites; the change was one of extended bearers and underlying technical improvements rather than extra headline functions.

The first product to use the m-Router based PC Connectivity was the Nokia 7650; its PC suite was a development of that from the 9210. Through Symbian OS v6.1, Symbian OS v7.0 and Symbian OS v7.0s, the PC Connectivity framework was improved in terms of performance and robustness but was not significantly extended. This is not to say that all the PC suites looked the same – the PC suites provided by Sony Ericsson with the P800 and P900 Symbian OS v7.0 smartphones are significantly different from those provided by Nokia. Other smartphone manufacturers have also tried different approaches to make their smartphones more attractive.

With the earliest Symbian OS smartphones, manufacturers were concerned with supplying an attractive and robust PC suite for each product, with a certain amount of innovation. When manufacturers released their second or third Symbian OS smartphones, they began to consider the value of standardization and moved towards developing PC suites to support a range of their smartphones rather than providing a different PC suite with each model of smartphone.

During 2002, Symbian introduced alternative PC software based around SCOM (Symbian Connect Object Model). SCOM was intended to require less resources when running than the previous PC software and was designed to provide standardized functionality across as many types of Symbian OS smartphone as possible. Subsequently, another layer of software called BAL (Bearer Abstraction Layer) was added to provide a standardized way of accessing connected phones and services. SCOM and BAL did not introduce any changes to any software on the smartphone – they used the existing protocols rather than adding new ones.

Since its creation, SCOM and BAL have been maintained to support as many Symbian OS smartphones as possible. At the time of writing, SCOM and BAL work with all the Symbian OS smartphones that use m-Router and the TCP/IP based services. It is impossible to guarantee this in the future as smartphone manufacturers continue to improve their products and some manufacturers will regard innovation as more important (that is, more attractive to consumers) than standardization. In some cases this will mean that individual models of Symbian OS smartphone will work well with SCOM, but will also have additional services, whereas in other cases the services may be so changed that SCOM and BAL may not work with them.

2.4 PC Connectivity Using OBEX

In the previous section I mentioned smartphone manufacturers seeking to standardize their PC suites. It is worth bearing in mind that all manufacturers of Symbian OS smartphones also make other mobile phones and smartphones, so there is an attraction to creating a PC suite that supports both Symbian OS smartphones and other, non-Symbian OS, smartphones.

However, most non-Symbian OS smartphones do not use TCP/IP to connect to Windows PCs. Instead, they use OBEX, which is well suited to exchanging small objects such as contact details and SMS messages and also supports larger objects such as file transfer.

Symbian OS includes some support for OBEX, but smartphone manufacturers have extended this and added services. This means that there is not currently a standard Symbian-supported PC Connectivity framework using OBEX, and for any development with such PC suites the developer will require support from the smartphone manufacturer.

3

An Architectural Overview of PC Connectivity

Most books on programming Symbian OS include a description of the architecture. This chapter describes the architecture of Symbian OS PC Connectivity without going into as much detail on the other Symbian OS internals.

Figure 3.1 is a deliberately simplistic view of PC Connectivity, but it shows the important features – a connection between the PC and the Symbian OS smartphone, a client application on the PC, and a server or service on the smartphone.

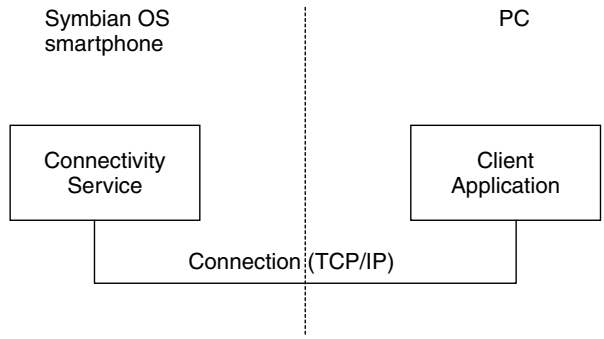

Figure 3.1 PC Connectivity

3.1 The Bearers, TCP/IP and PPP

All the PC Connectivity examples in this book run on Symbian OS smartphones that use a TCP/IP connection as described in the previous chapter. The TCP/IP connection between the relevant client and service runs over a PPP link that can be carried over RS232-serial, infrared, Bluetooth and USB. The use of PPP and TCP/IP to abstract the physical

bearers means that the rest of the PC Connectivity software (on both the PC and the smartphone) does not need to have any awareness of the actual bearer in use.

Although Microsoft Windows uses PPP to connect user PCs to servers, it does not include a PPP implementation that will natively run over infrared, Bluetooth and USB as well as RS-232 serial. That is why Symbian PC Connectivity uses m-Router to provide a PPP implementation.

The good news is that all the difficult problems (and handling the various Microsoft and third-party communications stacks can cause a range of problems) are hidden from the developer who is creating new PC Connectivity services.

3.2 A Client-Server Model of PC Connectivity

Given a TCP/IP connection, we need two more components – the software to run at each end of the connection. With a PC Connectivity application, one of the software components runs on the Symbian OS smartphone and the other runs on the PC.

The normal way of describing these components is to call the software running on the smartphone a service or a server and to call the software running on the PC a client. These names match the behavior of most PC Connectivity applications. It is possible to reverse these roles and have a client on the smartphone using a service on the PC, but this is less common and there is less support for it.

It is worth bearing in mind that this client-server model of PC Connectivity hides additional levels of complexity. Symbian OS uses the client-server model extensively as an internal pattern – servers are used to control access to shared resources throughout Symbian OS. Therefore, a PC Connectivity server or service running on the smartphone is almost certainly a client of one or more other servers on the smartphone.

In Chapters 4 and 5 we will use services that access the filing system on the smartphone; these use PC Connectivity services that make use of the Symbian OS file server. In Chapters 9–12 we will create further PC Connectivity services that make use of the Message server and the Contacts and Agenda servers.

Symbian OS provides a number of PC Connectivity services as standard in order to provide the functions expected from a PC suite. It is possible to access these directly from the PC, but this is not recommended for two reasons.

Firstly, Symbian has not published documentation for the protocols, and Symbian and the smartphone manufacturers have made improvements and alterations across the different versions of Symbian OS and even between different smartphone models based on one version of Symbian OS. Therefore, although it has been done, reverse engineering

the protocols is difficult and prone to unpredictable differences between smartphone models.

Secondly, Symbian provides a layer of middleware (called SCOM) described in Chapter 4 and used in Chapter 5 to access the standard services. SCOM has the task of handling protocol differences and also exposes an API that is much easier to use than driving the protocols directly.

If you want to create a new PC Connectivity service (as we will in later chapters) then you have a series of challenges:

1. You need to create the software on the Symbian OS smartphone to actually provide the service.
2. You need to create the software on the PC to use the service.
3. You need a way of starting the service on the smartphone when required (you do not want to have the service running when it is not required).
4. You need a way of establishing a connection between the PC software and the service.

Challenges 3 and 4 are addressed by the Symbian OS PC Connectivity framework. In Symbian OS v6.1 to Symbian OS v7.0s you can create the service on the smartphone as a custom server as described in Chapter 7. The PC Connectivity framework then provides methods to load the server and connect to it. In Symbian OS v8.0 onwards the custom server APIs are replaced by socket server APIs described in Chapter 8. In both of these cases I will show the commands required to use your services.

As an alternative, it is possible to create your service as a standard TCP/IP socket server that knows nothing about PC Connectivity. In this case the challenge is to start the service and connect to it, and this is covered in Chapter 14.

4

The Symbian Connect Object Model

4.1 Overview

SCOM (the Symbian Connect Object Model – pronounced 'escom') is a reusable software component that allows developers to more easily produce applications that incorporate connectivity with Symbian OS smartphones. While SCOM does this by abstracting core connectivity features, it also provides the ability for developers to access other services on the phone which may be developed either by themselves or by a third party. SCOM is an out-of-process COM server that supports multiple clients. SCOM is not an application that can be directly used by an end-user. Instead, some form of application must be created that uses SCOM in a way that is helpful to the end-user. This chapter describes the functionality provided by SCOM that can be used by an application.

4.2 Functionality in SCOM and in PC Suites

SCOM does not provide all the functions that a user might expect. It provides functionality to manage device connections and services, and it provides simple access to some core services that Symbian considers should be common to all Symbian OS smartphones.

SCOM was originally created with the needs of smartphone manufacturers in mind. These Symbian licensees have to provide a PC suite to accompany their smartphones. Typically, the suites include the following functionality:

- backup – copying files that include data, settings and installed applications from the smartphone to the PC
- restore – restoring the files that include data, settings and installed applications back to the smartphone to restore it to a previous state
- installation of new software (Symbian OS applications or Java applications) on the smartphone

- synchronization with PIMs on the PC to keep contacts and calendar data up to date
- some form of image or sound file management that requires the ability to copy files to and from the smartphone.

Often, Symbian OS smartphone manufacturers will provide additional functionality in order to give their smartphones a competitive advantage, but Symbian cannot predict this functionality and so SCOM cannot directly support it (although it does provide the means to access any additional services by means of stream interfaces).

SCOM provides the following functionality directly:

- backup and restore
- file management
- software install.

PIM synchronization is not directly provided by SCOM. It, instead, allows specialized synchronization software access to services on the Symbian OS smartphone.

It is possible for any developer with sufficient skill and resources to create a complete PC suite based on SCOM, but Symbian does not regard that as sensible. You should assume that the smartphone manufacturers or specialist partners will create their own PC suites and that it will not be sensible to compete directly with them. Instead, developers should focus their attentions on creating applications that complement the PC suites provided with the smartphones. Therefore, this book shows how to carry out functions useful to third-party developers and does not attempt to show how to create a full PC suite. The backup and restore and software installation functionality are omitted from this chapter, and the protocols used for them and for PIM synchronization are not covered.

The main areas of functionality covered by this book are:

- managing connections to devices and starting and using services on Symbian OS smartphones
- file management functions on Symbian OS smartphones.

4.3 SCOM and BAL

SCOM is the higher-level API provided to manipulate devices and their filing systems. BAL (the Bearer Abstraction Layer) is a slightly lower-level API that manages device connection and disconnection and services on the device. SCOM uses BAL to provide its own API (take a look at the

respective device properties and the mapping will become apparent). It is possible to use SCOM without making direct use of BAL – indeed, this is how SCOM was originally intended to be used. However, the BAL service API is slightly more efficient, in terms of performance, than that of SCOM and so the developer may choose to use BAL for some operations. In later chapters we will start up services on the phone, using SCOM and BAL, which are then accessed by means of Windows sockets in various guises.

Figure 4.1 gives a simplified view of the components that a PC Connect application interacts with.

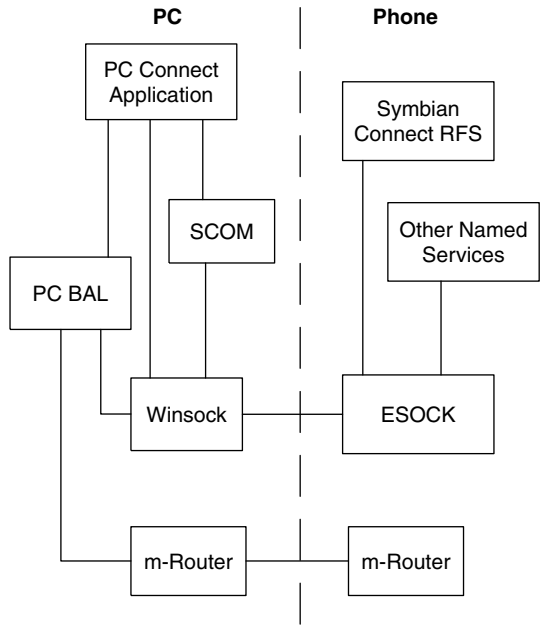

Figure 4.1 SCOM and BAL

4.4 COM Programming and Language Choice

SCOM and BAL are built as COM servers because this provides the simplest way to access their functionality. It also allows SCOM and BAL to be used from any COM-compatible language and so makes them available to the widest possible set of developers. On a PC this means that just about any development language can access SCOM and BAL.

This book does not attempt to provide an introduction to COM – the bookstores have shelves of books on COM (and all its variants) and it would be wasteful to reproduce their contents here. This book does include specific guidance on using SCOM and BAL in several languages,

so you will be able to use SCOM and BAL even if you have never used COM.

Similarly, this book is not intended as a tutorial in C#, C++ or Visual Basic because there are even more books on these subjects than on COM. The examples in the subsequent chapters are mostly written in C# and the logic should be apparent to any developer, although the way COM is used in different languages means that the actual class and method names will vary slightly. Sufficient examples are provided in Visual C++ and in VB to get developers started in those languages; standard IDE tools will then allow you to obtain the detail that you need on class and method names.

4.5 Error Handling

All the methods provided by SCOM and BAL can fail with bad `HRESULT`s and developers should check the return values. In C# these errors are thrown as exceptions that must be caught.

SCOM provides rich error information to clients by means of `IErrorInfo`, but these error descriptions are not localized. Therefore, they must not be displayed to the user – they are intended just as debugging aids.

4.6 SCOM Class Reference

This section lists the classes that make up SCOM and BAL that are intended for use by third-party developers and describes the API for these classes. It omits some classes and APIs that are intended only for use by smartphone manufacturers (these can be accessed using the type libraries, but I suggest that you ignore them).

You will see that some of the class names are of the form <name><number>, for example `ISCDevice2`. These are classes that have been extended. SCOM developers follow the rules for COM development and so, once the interface to a class has been defined and published, they will not change it. However, there have been cases where additional functionality has been desirable and so classes have been extended by defining a new class that replaces the old one. In such cases you should normally use the 'latest' class to have access to the most functionality.

All the classes and types listed are part of the `SymbianConnect` namespace. The types of members and arguments are given in C# terminology; it should be straightforward to convert these to the appropriate types for C++ or VB. In any case, the type libraries will provide information on the types in a language-specific form.

Figure 4.2 is a simplified view of the major SCOM classes and interfaces that you will use.

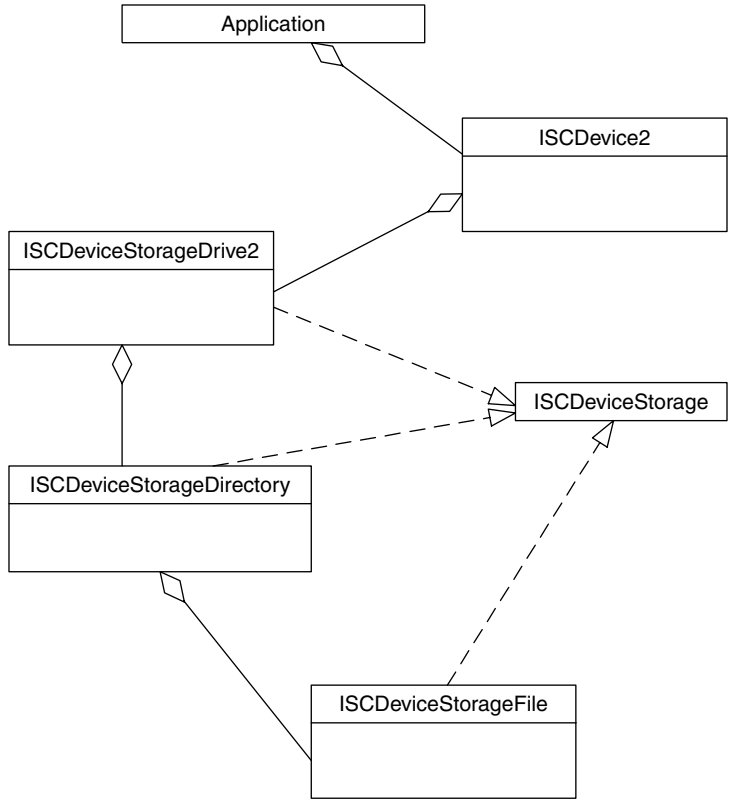

Figure 4.2 SCOM Classes

4.6.1 SCOM Application, Connection and Device Classes

These classes handle an SCOM application (which provides access to connected devices and provides a basis for event handlers) and connected devices.

Class `SymbianConnect.Application` – This class implements `ISCApplication` and `ISCEvents` and provides access to connected devices and to SCOM events.
Member Variables
`ISCDeviceCollection ConnectedDevices` This read-only member provides access to the set of currently connected devices. See also `DeviceConnected` and `DeviceDisconnected` events.
Event Handlers (part of `ISCEvents` interface – described in Section 4.6.3)

`DeviceConnected` This event handler is called when a new device is connected. See `ISCEvents_DeviceConnectedEventHandler`.
`DeviceDisconnected` This event handler is called when a device is disconnected. See `ISCEvents_DeviceDisconnectedEventHandler`.
`DeviceCopyStorageFileProgress` This event handler is called to report progress during a file copy operation. See `ISCEvents_DeviceCopyStorageFileProgressEventHandler`.
`DeviceCopyStorageFileError` This event handler is called when an error occurs during a file copy operation. See `ISCEvents_DeviceCopyStorageFileErrorEventHandler`.
`DeviceCopyStorageFileExistingFileFound` This event handler is called when a file copy operation encounters an existing file with the same name as the target. See `ISCEvents_DeviceCopyStorageFileExistingFileFoundEventHandler`.
`DeviceCopyStorageFileComplete` This event handler is called when a file copy operation is complete. See `ISCEvents_DeviceCopyStorageFileCompleteEventHandler`.

Class `ISCDevice2` – This interface provides access to the properties of a connected device and its storage.
Member Variables
`string ConnectionBearer` This read-only member provides a string that describes the transport used to connect the device.
`string Manufacturer` This read-only member provides the name of the manufacturer of the device.
`string Model` This read-only member provides the name of the model of the device. This should be used in conjunction with the manufacturer.
`string Id` This read-only member provides a unique identifier for the device. It is guaranteed to be different from any other device and can be used to identify it. Commonly, it is the IMEI number of the smartphone.

`ISCDeviceStorageDriveCollection StorageDrives`
This read-only member provides the set of drives owned by the device. This member is the root for navigation of the filing system on the device.

`ISCDeviceStorage Storage [string]`
This read-only member is used to directly access a directory or file on the device. It is possible to access any file or directory on the device by navigating through the directory tree, but this can be tedious. Instead, this member is used with the full path of the file or directory required (directories should be terminated with a trailing backslash, `\`) and returns an object that can be cast to an `ISCDeviceStorageDirectory` or `ISCDeviceStorageFile`.

`bool IsActive [string]`
This read-only member is obsolete and should not be used. It is used by some legacy synchronization software.

Member Methods

`void SynchroniseDateTime ()`
This method synchronizes the device date and time with that of the PC. It is commonly used as a part of synchronization and backup operations.

`ISCSequentialStream OpenDeviceService (string aServiceName)`
This method attempts to open a service on the device by name and returns a stream that can be used to communicate with the service. This is how SCOM provides access to lower-level or third-party services on the smartphone.
`aServiceName` – the name of the service to be started on the device.
returns – a stream object if successful or null if unsuccessful.

`SCAsyncStreamSink OpenAsyncDeviceService(string aServicename)`
This method attempts to open a service on the device by name and returns a stream that can be used to communicate with the service asynchronously. This is how SCOM provides asynchronous access to lower-level or third-party services on the smartphone.
`aServiceName` – the name of the service to be started on the device.
returns – an asynchronous stream object if successful or null if unsuccessful.

`void SetActive(string)`
This method sets a device as the active device. This concept is used only by some legacy synchronization software and should not be used elsewhere.

Class `ISCDeviceCollection`

This class is a collection of `ISCDevice` objects. It can be accessed by standard iterators. Note that its first index is 1, not 0.

4.6.2 SCOM Storage Classes

Enumerated Type `ScStorageType`

- `scDrive`
- `scDirectory`
- `scFile`

Class `ISCDeviceStorage` – This interface is the base for drives, directories and files.

Member Variables

`ScStorageType Type`
This read-only property gives the type of the device storage object.

`string Path`
This read-only property gives the path of the device storage object.
Note that paths always start with a drive letter and a colon, directory paths always terminate with a backslash (`\`), and file paths never terminate with a backslash.

Flag Type `ScDriveAttributes`

Not all of these attributes may be applicable to a smartphone.

- `scDriveAttLocal = 0×01`
- `scDriveAttROM = 0×02`
- `scDriveAttRedirected = 0×04`
- `scDriveAttSubsted = 0×08`
- `scDriveAttInternal = 0×10`
- `scDriveAttRemovable = 0×20`
- `scDriveAttRemote = 0×40`
- `scDriveAttTransaction = 0×80`

Enumerated Type `ScDriveBatteryState`

- `scBatteryGood`

- scBatteryLow
- scBatteryNotSupported

Flag Type `ScMediaAttributes`

• scMediaAttVariableSize	=	0×01
• scMediaAttDualDensity	=	0×02
• scMediaAttFormattable	=	0×04
• scMediaAttWriteProtected	=	0×08
• scMediaAttLockable	=	0×10
• scMediaAttLocked	=	0×20
• scMediaAttHasPassword	=	0×40

Enumerated Type `ScMediaType`

- scMediaCdRom
- scMediaFlash
- scMediaFloppy
- scMediaHardDisk
- scMediaNotPresent
- scMediaRam
- scMediaRemote
- scMediaRom
- scMediaUnknown

Class `ISCDeviceStorageDrive2` – This interface provides access to the properties of a drive and access to the directories on the drive.

Member Variables

`int Attributes`
This read-only property is a combination of zero or more `ScDriveAttributes` flags.
This property does not always provide meaningful values, so test it with specific devices.

`ScDriveBatteryState BatteryState` This read-only property indicates whether or not the drive supports a battery and, if so, its state. This property does not always provide meaningful values, so test it with specific devices.
`int Capacity High` This read-only property is the high 32-bits of the drive capacity in bytes. Because of COM automation compatibility, this is returned as a signed value and must be cast to an unsigned value before combining it with the other half of the value.
`int CapacityLow` This read-only property is the low 32-bits of the drive capacity in bytes. Because of COM automation compatibility, this is returned as a signed value and must be cast to an unsigned value before combining it with the other half of the value.
`int FreeSpace High` This read-only property is the high 32-bits of the drive free-space in bytes. Because of COM automation compatibility, this is returned as a signed value and must be cast to an unsigned value before combining it with the other half of the value.
`int FreeSpaceLow` This read-only property is the low 32-bits of the drive free-space in bytes. Because of COM automation compatibility, this is returned as a signed value and must be cast to an unsigned value before combining it with the other half of the value.
`long MediaAttributes` This read-only property is a combination of zero or more `ScMediaAttributes` flags. This property does not always provide meaningful values, so test it with specific devices.
`ScMediaType MediaType` This read-only property gives the type of media mounted on the drive. This property does not always provide meaningful values, so test it with specific devices.
`string Path` This read-only property gives the path of the device storage object.
`ISCDeviceStorageDirectory RootDirectory` This read-only property gives the directory at the root of the drive.
`ScStorageType Type` This read-only property gives the type of the device storage object.
`int UniqueId` This read-only property gives the unique identifier for the drive. This is meaningful only for some types of removable drive and it changes after formatting with some devices.
`string VolumeLabel` This writable property gives the volume label of the drive. Setting this property will set the volume label of the drive.

Member Methods

`int Format ()`
This asynchronous method initiates a format operation on the drive. The return value is the request ID that will be returned by event handlers. Please note that SCOM may not be able to format the `c:` drive because it may contain files which are necessary to maintain the connection. It is probably unwise to try to format the `c:` drive anyway, because it contains essential data.

`void Refresh ()`
SCOM caches information about drives, directories and files to provide fast access to that information, as it would be slow to retrieve that information on demand whenever required. This method refreshes that stored information about a drive. It is likely to be most useful if the drive is a removable drive such as an MMC card.

Class `ISCDeviceStorageDriveCollection`

This class is a collection of `ISCDeviceStorageDrive` objects. It can be accessed by standard iterators.
Note that its first index is 1, not 0.

Class `ISCDeviceStorageDirectory` – This interface provides access to the properties of a directory and the child directories and files that it owns, and supports a range of operations on the directory.

Member Variables

`ScStorageType Type`
This read-only property gives the type of the device storage object.

`string Path`
This read-only property gives the path of the device storage object.

`ISCDeviceStorage Parent`
This read-only property gives the parent of the device storage object. This will be an `ISCDeviceStorageDirectory` for all directories except root directories for which it will be an `ISCDeviceStorageDrive`.

`ISCDeviceStorageDirectoryCollection ChildDirectories`
This read-only property gives the collection of child directories.

`ISCDeviceStorageFileCollection ChildFiles`
This read-only property gives the collection of files in the directory.

Member Methods

`int CopyFileFromPC(string aFileToCopy)`
This asynchronous method copies a file from the PC to the device.
`aFileToCopy` – the name of the file on the PC to copy to the directory.
returns – the request ID for the file copy operation. This will be returned by subsequent file copy events.

`void Rename(string aNewName)`
This method attempts to rename the directory.
`aNewName` – the new name for the directory. This can be either a fully qualified directory name on the same drive ending with a backslash or an unqualified valid directory name in the same parent directory.

`void Delete()`
This method attempts to delete the directory. If the directory is not empty it can still be deleted – in fact SCOM will recursively delete all child directories and files and then delete the empty directory. This method should be used with care.

Class `ISCDeviceStorageDirectoryCollection`

This class is a collection of `ISCDeviceStorageDirectory` objects. It can be accessed by standard iterators.
Note that its first index is 1, not 0.

Class `ISCDeviceStorageFile` – This interface provides access to the properties of a file and supports a range of operations on the file.

Member Variables

`ScStorageType Type`
This read-only property gives the type of the device storage object.

`string Path`
This read-only property gives the path of the device storage object.

`ISCDeviceStorage Parent`
This read-only property gives the parent of the device storage object. This will be an `ISCDeviceStorageDirectory` for all files.

`string FileName`
This read-only property is the name of the file. Although it is read-only as a property, it can be altered using the `Rename()` method.

`int Size`
This read-only property is the size of the file.