

About this Book

The Accredited Symbian Developer (ASD) Examination is fundamentally based on the content of existing Symbian Press books, with the C++ curriculum deriving from general C++ literature. Arguably, if you own the core Symbian Press titles and a solid C++ reference book, you will have all the information you require to get through the exam. So why do you need an ASD Primer?

The problem is that there is a lot to know! Although the exam objectives are published, it can still be unclear what is really relevant and what is not.

To address this, the Primer is a pragmatic restructuring of a number of existing Symbian Press titles, primarily Jo's *Symbian OS Explained: Effective C++ Programming for Smartphones*. It also borrows from Richard Harrison's *Symbian OS C++ for Mobile Phones, Volume 2* and Steve Babin's *Developing Software for Symbian OS*. Where necessary, material from those books has been updated to meet the needs of an audience working with Symbian OS v9.1 and beyond. To do this, we took guidance on Platform Security and EKA2 from Craig Heath's *Symbian OS Platform Security* and Jane Sales' *Symbian OS Internals*, respectively.

We have organized the information in such a way that the reader can dip in and out of sections to refresh their knowledge and understanding of the fundamental concepts of Symbian software development. In this sense, the Primer can also serve as a desk reference for both Symbian OS programming and, more modestly, C++, as well as a revision guide for ASD exam candidates.

The C++ sections have focused on information in Stroustrup's essential *The C++ Programming Language* as the primary reference, but we also found ourselves returning to Scott Meyer's invaluable *Effective C++* and

Stephen Dewhurst's excellent *C++ Common Knowledge*. Please see the References section for more information.

Our intent was not to rehash material but to provide every ASD candidate with a solid foundation in the subject matter, so that they can enter the exam with confidence.

That said, simply reading the book, or even having it present when you sit the exam, will not guarantee you a pass, because you will still need to understand the subject matter. The best classroom for this is the workplace or a study project, where the lessons learnt in this book can be put into practice. If you understand the contents of this Primer and have about two years' exposure to Symbian OS, you should have the requisite skills to pass with flying colors and become an ASD.

Example ASD questions and the sample code for this book are available for download from the Symbian Press website, **www.symbian.com/developer/books**, and from Meme Education, **www.meme-education.com**.

We hope you enjoy the journey towards Accreditation!

Jo Stichbury
Mark Jacobs
Summer 2006

About the Authors

Jo Stichbury

Jo Stichbury was educated at Magdalene College, Cambridge, where she held the Stothert Bye-Fellowship. She has an MA in Natural Sciences and a PhD in the chemistry of Organometallic Molybdenum complexes. She has worked within the Symbian ecosystem since 1997 in the Base, Connectivity and Security teams of Symbian. She has also worked for Advansys, Sony Ericsson and Nokia. Jo became an Accredited Symbian Developer in 2005 and a Forum Nokia Champion in 2006.

Jo is also the sole author of *Symbian OS Explained: Effective C++ Programming for Smartphones* which was published by Symbian Press in 2004. She fled the UK for Canada shortly before it was released, but was tracked down in Vancouver and pressed back into service to co-author this new book about the Accredited Symbian Developer exams.

Mark Jacobs

Mark wrote his first computer program in early 1980. He wrote his first C++ program in 1987, in an effort to prove that Modula-2 was the better language, and has enjoyed an ongoing bipolar relationship with the language ever since.

He joined Symbian in January 2000, stage diving into the last few months of development of Symbian OS v6.0, which was not totally dissimilar to a Motorhead back-stage party hosted by NASA. For the

majority of his time at Symbian, Mark was a System Architect. He left London for Vancouver in 2004 and founded Meme Education in late 2005.

He has a BSc in Computer Science from the University of Hertfordshire and is an Accredited Symbian Developer. He lives with Jo, two Siamese cats and a La Pavoni espresso machine.

Acknowledgments

First and foremost, a big thank you to Coach Freddie (Sven) Gjertsen of Symbian Press. Freddie has been a complete star: listened to our rants, bribed us, and just got stuff done.

No Symbian Press book would be complete without a mention of Phil Northam, who came to Vancouver on what can only be described as a 5500 mile, first-class, extended pub crawl of the Piccadilly Line, to buy us lunch and to convince Jo that she desperately needed to write another book. Thanks also to Drew Kennerly and all at John Wiley for making things worryingly easy, and to Satu McNabb of Symbian Press.

A note of thanks goes to our fellow Symbian Press authors: Richard Harrison, Steve Babin, Craig Heath, Jane Sales (and Boris and Mishka), Michael Jipping and all those who worked on the fine Symbian Press titles that we drew upon while writing this one.

We'd also like to thank the reviewers who contributed enormously to the technical accuracy of this book.

Jo would like to thank her colleagues for their support and understanding while she took time out to write this book, in particular Van Ly and Jon Bruce, Bill Bonney, Amonn Phillip and Kevin Chan. Thank you for your patience; normal service should be resumed once I figure out where I left it.

A special thanks goes to Ian Weston of Majinate, whose integrity, advice and just plain "doing the right thing" approach has been invaluable to us. A nod of respect also goes to the good folk of Symsource. And, of course all those who wrote the exam. Thanks, chaps – they still hurt even when you know the answers.

This book was written using Pages 2. Thanks to Apple Computer, Inc. for giving us an alternative.

Peace and love to James, Yvonne, Viv, Val, Nicky, Clive, and Scott.

Symbian Press Acknowledgements

We would like to thank Mark and Jo for the long hours that they dedicated to the creation of this book and hope that they are not too emasculated by their endeavors.

Thanks are also due to all our captious reviewers, named and otherwise, for the insights that they produced, to deadline.

Introduction

by Ian Weston, Majinate

Software development on the Symbian platform is a delicate discipline, requiring an appreciation of a wide range of issues that do not necessarily apply to other development environments. Symbian OS, since its creation, has been specifically developed for mobile phones, and as a consequence it uses different development concepts from that of standard desktop or server development. These concepts are designed to:

- provide efficient power management
- ensure fast boot times
- allow phones to run indefinitely without rebooting
- conserve and recycle memory
- allow software installation without restarting the operating system
- make use of the permanent connectedness of networked devices
- make phones robust when networks and connections fail.

What Is the Accredited Symbian Developer Scheme?

The balance of opportunity and constraints is clearly different for mobile software development, and a software developer who appreciates and understands these Symbian OS concepts is a highly skilled individual. The Accredited Symbian Developer (ASD) qualification allows professional Symbian developers to validate their understanding and knowledge of Symbian OS software development with an industry-recognized certificate of professional achievement. The goal of the ASD scheme is to ensure

that the Symbian ecosystem is provided with better-educated developers, and to allow those developers to differentiate themselves within the ecosystem.

The ASD scheme publishes the “ASD Curriculum”, the set of knowledge required to understand and develop software for Symbian OS. The scheme, and especially the curriculum, provides a framework for continuing professional development, keeping pace with the evolution of Symbian OS. The curriculum is updated as new features are introduced into the operating system and old features are deprecated. This is done in a measured manner with full visibility of the future use of Symbian OS in mobile phones and with knowledge of the long-term evolution of the system.

The curriculum ensures that exam candidates are tested objectively, on material that is readily accessible. The exam is able to differentiate between levels of proficiency to the extent that candidates above a threshold of competence, independently defined and reviewed by Symbian, are assigned the status “Accredited Symbian Developer”.

The ASD infrastructure provides an excellent tool for unbiased, non-threatening testing of competency and understanding of the curriculum. The option of a detailed proficiency analysis allows training managers to target training investment appropriately and assess the return accurately through a sequence of exams giving measurable improvements in performance. In turn, this means that employers can better allocate engineering staff to appropriate tasks and roles. The full examination can also be used to obtain interview preparation sheets by generating a report of the relative strengths and weaknesses of the candidate across the curriculum topics. This is highly valuable in interview screening, giving more effective use of face-to-face interview time and avoiding the need for longer technical interviews.

The Accredited Symbian Developer Exam in Detail

The examination infrastructure is web-based and supplied by Majinate Limited, enabling the exam to be sat anywhere in the world with an internet connection by visiting www.majinate.com/takeexam.

The exam questions test the candidate’s knowledge of each of the ASD curriculum areas. These questions range in difficulty from straightforward (requiring little real-world exposure and limited book study) through to devilishly hard (requiring years of experience and a thorough understanding of the principles of the operating system). During any examination session, an adaptive mechanism, which takes account of how well the previous questions on a subject were answered, is used to determine the difficulty of the next question. This ensures that each candidate is presented with questions that test to the limits of his or her understanding.

The examination does not have a predetermined number of questions nor duration; however, the majority of candidates can expect to complete the test in 70 minutes and spend approximately 90 seconds on each question.

Each question in the exam is presented as a statement with five answers. Up to three of these answers may be correct. The candidate is required to mark as many of the correct answers as possible and avoid marking any wrong answers (which may be there to distract the unwary). Only by selecting all the correct answers in the fastest time will the maximum number of marks for a question be achieved. Negative marks are awarded for slowness, for failing to select a right answer, or for selecting a wrong one. Skipped questions (those where no answers are chosen by the candidate) are marked conservatively, and a single choice of answer which is wrong may lead to a better final mark than a skipped question. In deciding how to answer a question, therefore, candidates should try to choose at least one answer which is likely to be right.

Exam Results and Controls

At the end of the exam, the results are analyzed offline and compared with the Symbian-defined pass mark. Within 48 hours, the candidate receives an email with the exam result. Candidates who pass are informed of their certificate number and when the certificate will be delivered to them. Candidates who do not demonstrate the required standard are advised of the areas where they were weakest as a recommendation for further study. There is no cooling-off period, and a candidate is free to re-sit the exam at any time. There is no consideration of the earlier exam in the presentation, analysis or results of later attempts.

In order to determine the level a candidate must attain to be recognized as an ASD, the initial pass mark was determined by examining a large contingent of Symbian and Symbian Partner engineers. The pass mark is reviewed annually to ensure that it maintains the high standards of the ASD scheme.

There are built-in security features to detect anomalous behavior that might indicate cheating, and the system reacts accordingly. In order to maximize the accessibility of the exam, it can be run in two modes: for candidates who are unable or unwilling to attend a testing centre supervised by one of Majinate's partner companies worldwide, it is possible to sit the exam in self-supervised mode. Self-supervised candidates sit exactly the same exam as supervised candidates (other than the exam environment) and receive a certificate indicating that the exam was not taken under supervision. Majinate is unable to confirm that a person holding such a certificate did indeed sit the exam unaided. For supervised candidates this is guaranteed by the supervisor, whose details are supplied with the examination certificate.

Potential employers and interviewers are able to run a short form of the exam online and on demand, with results available immediately. This, although not sufficient to repeat the examination exercise, confirms whether the candidate falls into the correct percentile as claimed by the certificate.

Exam Essentials Summary

1 C++ Language Fundamentals

1.1 Types

- Understand that C++ has a number of different types in various categories for example integral, arithmetic
- Recognize `typedef` as defining a synonym for an existing type but not a new type in itself
- Recognize enumeration types as user-defined value sets
- Specify the advantages of `const` over `#define`
- Understand the use and properties of the C++ reference type
- Specify the difference between pointers and references
- Understand the semantics of pointer arithmetic
- Recognize pointer operations and the purpose of the `NULL` pointer value
- Differentiate `const` pointers and pointers to `const`

1.2 Statements

- Know the use and properties of the declaration and definition statements
- Recognize the use of the `extern` keyword

- Cite and understand initialization of variables and their scope
- Understand the purpose, syntax and behavior of C++ loop statements (`while`, `for` and `do`)
- Specify the behavior and effect of the `continue` and `break` keywords in a loop
- Specify the syntax and behavior of C++ conditional statements (`if` and `switch`)

1.3 Expressions and Operators

- Specify the syntax and meaning of unary and binary operator expressions
- Understand the difference between precedence and associativity
- Recognize common operator categories including logical, prefix and postfix
- Demonstrate awareness of general operator precedence and associativity rules

1.4 Functions

- Understand the syntax of a function prototype
- Cite the purpose of the `inline` keyword
- Understand the rules for passing default arguments and an unspecified number of arguments to a function
- Recognize value, reference, array and pointer parameter passing
- Understand the scope of function blocks and return by reference and value
- Specify the syntax for pointers to function assignments
- Recognize pointer to functions as callback parameter arguments

1.5 Dynamic Memory Allocation

- Understand C++ free store allocation scope using the `new` and `delete` operators
- Recognize the syntax and purpose of placement `new`

1.6 Tool Chain Basics

- Understand the function of the tools, the C++ tool chain (for example compiler and linker)
- Recognize the lexical and syntax-parsing stages of compilation
- Be able to describe the purpose of the C++ preprocessor, specifying common directives
- Understand the role `inline` functions play in C++
- Know how to use the `extern` keyword

2 Classes and Objects

2.1 Scope and C++ Object-Oriented Programming (OOP) Support

- Understand the scope and lifetime properties of blocks and namespaces
- Understand C++ support for data abstraction
- Specify the attributes of a C++ object with respect to object-oriented programming
- Know the syntax of a class declaration
- Cite the differences between a class and an object
- Differentiate between basic data structures (`struct` keyword) and classes

2.2 Constructors and Destructors

- Understand the order of construction and destruction for the class and its member variables
- Recognize implicit constructor invocation (overloading and pattern matching) and the role the `explicit` keyword plays in constructor declaration
- Specify what the compiler automatically generates for user-defined classes
- Understand the purpose and use of copy constructors (including parameter passing)
- Understand the difference between assignment and initialization

- Specify the required function members needed in a class to support safe ownership of pointer data

2.3 Class Members

- Describe `private`, `protected` and `public` access control for class members
- Declare and specify the syntax for pointers to class members
- Identify nested classes and their scope and lifespan
- Cite the scope access rules and lifetime of a nested class
- Understand the scope rules and syntax of `friend` functions
- Understand the semantics of member functions and data addressing
- Understand the purpose of the scope-resolution operator `::`
- Understand the role of the `this` pointer
- Specify the properties of `static` class members

3 Class Design and Inheritance

3.1 Class Relationships

- Understand the key benefits and purpose of inheritance
- Specify the differences between composition, aggregation and inheritance
- Cite the object-oriented relationship for inheritance

3.2 Inheritance

- Be able to define public inheritance
- Understand the scope resolution operator syntax for accessing the base-derived class hierarchy
- Given a base-derived hierarchy, specify the access rules (including those of friend classes)
- Describe the scope access rules and purpose of public, protected and private inheritance
- Specify the implicit invocation order of constructors and destructors in a base-derived hierarchy

3.3 Dynamic Polymorphism – Virtual Methods

- Specify the mechanisms of OO reuse available in C++
- Be able to state C++ support for polymorphism
- Understand the purpose of and difference between overriding and overloading
- Understand the use of overriding to modify behavior in base-derived class inheritance
- Understand the rules and pattern-matching criteria for correct overloaded-function invocation
- Identify the typical uses and behavior for operator overloading
- Describe the purpose of the virtual table, citing constraints and overheads
- Specify the use of virtual functions and their implementation tradeoffs
- Understand how an abstract base class is implemented in C++
- State the differences between interface and implementation inheritance
- Understand and recognize the problems associated with multiple inheritance
- Cite the implementation requirements to support the `static_cast` operator in user-defined classes

3.4 Static Polymorphism and Templates

- Specify the syntax for a simple function template specialization
- Be able to cite the advantages of function templates (for example over macros)
- Understand the inheritance rules and syntax supported by class templates
- Understand the syntax and semantics of a template type/class declaration
- Recognize the prototype declaration and pattern-matching properties of a template declaration and its use
- Understand the purpose and implementation differences of the Symbian OS thin template and mainstream C++ templates

4 Symbian OS Types and Declarations

4.1 The Fundamental Symbian OS Types

- Know how the fundamental Symbian OS types relate to native built-in C++ types
- Understand that the fundamental types should always be used in preference to the native built-in C++ types (`bool`, `int`, `float`, etc.) because they are compiler-independent

4.2 T Classes

- Know the purpose of a T class, what types of member data it may and may not own, and that it must never have a destructor
- Know what types of function a T class may have
- Understand that a T class may be created on the heap or stack
- Understand that a T class may be used as an alternative to the traditional C/C++ `struct`
- Know that the T prefix is also used to define an `enum`.

4.3 C Classes

- Recognize that a C class always derives from `CBase`
- Know the purpose of a C class, and what types of data it may own
- Understand that a C class must always be instantiated on the heap
- Know that a C class uses two-phase construction and has its member data zero-filled when it is allocated on the heap
- Understand the destruction of C classes via the virtual destructor defined in `CBase`

4.4 R Classes

- Know the purpose of an R class, to own a resource
- Understand that an R class can be instantiated on the heap or the stack
- Understand the separate construction and initialization of R classes
- Understand the separate cleanup and destruction of R classes, and the consequences of forgetting to call the `Close()` or `Reset()` method before destruction

4.5 M Classes

- Know the purpose of an M class, to define an interface
- Understand the use of M classes for multiple inheritance, and the order in which to derive an implementation class from C and M classes
- Know that an M class should never contain member data and does not have constructors
- Know what types of function an M class may include and the circumstances where it is appropriate to define their implementation
- Understand that an M class cannot be instantiated

4.6 Static Classes

- Know that static classes do not have a prefix letter
- Understand that static classes cannot be instantiated because they contain only static functions

4.7 Factors to Consider when Creating a Symbian OS Class

- Know the important factors to consider when creating a new class, and how this determines the choice of Symbian OS class type

4.8 Why Is the Symbian OS Naming Convention Important?

- Understand that the use of a class prefix makes it clear to anyone wishing to use a class how it should be instantiated, used and destroyed safely.
- Recognize that the naming convention forces a class designer to think about the factors described in Section 4.7 and, having decided on the fundamental behavior, can concentrate on the role of the class, knowing that leave-safe construction, destruction and ownership are already handled.

5 Leaves and the Cleanup Stack

5.1 Leaves: Lightweight Exceptions for Symbian OS

- Know that, before v9, Symbian OS does not support standard C++ exceptions (`try/catch/throw`) but uses a lightweight alternative: `TRAP` and `leave`

- Know that leaves are a fundamental part of Symbian error handling and are used throughout the system
- Understand the similarity between leaves and the `setjmp/longjmp` declarations in C
- Recognize the typical system functions which may cause a leave, including the `User::LeaveXXX()` functions and `new(ELeave)`
- Be able to list typical circumstances which cause a leave (for example, insufficient memory for a heap allocation)
- Understand that `new(ELeave)` guarantees that the pointer return value will always be valid if a leave has not occurred

5.2 How to Work with Leaves

- Know that leaves are indicated by use of a trailing L suffix on functions containing code which may leave (for example, `InitializeL()`)
- Be able to spot functions which are not leave-safe and those which are
- Understand that leaves are used for error handling; code should very rarely both return an error and be able to leave
- Understand the reason why a leave should not occur in a constructor or destructor

5.3 Comparing Leaves and Panics

- Understand the difference between a leave and a panic
- Recognize that panics come about through assertion failures, which should be used to flag programming errors during development
- Recognize that a leave should not be used to direct normal code logic

5.4 What Is a TRAP?

- Recognize the characteristics of a TRAP handler
- Understand that, for efficiency, use of TRAPs should be kept to a minimum

5.5 The Cleanup Stack

- Know how to use the cleanup stack to make code leave-safe, so memory is not leaked in the event of a leave

- Understand that `CleanupStack::PushL()` will not leak memory even if it leaves
- Know the order in which to remove items from the cleanup stack, and how to use `CleanupStack::PopAndDestroy()` and `CleanupStack::Pop()`
- Recognize correct and incorrect use of the cleanup stack
- Understand the consequences of putting a C class on the cleanup stack if it does not derive from `CBase`
- Know how to use `CleanupStack::PushL()` and `CleanupXXXPushL()` for objects of C, R, M and T classes and `CleanupArrayDeletePushL()` for C++ arrays
- Understand the meaning of the Symbian OS function suffixes C and D

5.6 Detecting Memory Leaks

- Recognize the use of the `__UHEAP_MARK` and `__UHEAP_MARKEND` macros to detect memory leaks

6 Two-Phase Construction and Object Destruction

6.1 Two-Phase Construction

- Know why code should not leave inside a constructor
- Recognize that two-phase construction is used to avoid the accidental creation of objects with undefined state
- Understand that constructors and second-phase `ConstructL()` methods are given private or protected access specifiers in classes which use two-phase construction, to prevent their inadvertent use
- Understand how to implement two-phase construction, and how to construct an object which derives from a base class which also uses a two-phase method of initialization
- Know the Symbian OS types (C classes) which typically use two-phase construction

6.2 Object Destruction

- Know that it is neither efficient nor necessary to set a pointer to `NULL` after deleting it in destructor code

- Understand that a destructor must check before dereferencing a pointer in case it is `NULL`, but need not check if simply calling `delete` on that pointer

7 Descriptors

7.1 Features of Symbian OS Descriptors

- Understand that Symbian OS descriptors may contain text or binary data
- Know that descriptors may be narrow (8-bit), wide (16-bit) or neutral (which is 16-bit since Symbian OS is built for Unicode)
- Understand that descriptors do not dynamically extend the data area they reference, so will panic if too small to store data resulting from a method call

7.2 The Symbian OS Descriptor Classes

- Know the characteristics of the `TDesC`, `TDes`, `TBufC`, `TBuf`, `TPtrC`, `TPtr`, `RBuf` and `HBufC` descriptor classes
- Understand that the descriptor base classes `TDesC` and `TDes` implement all generic descriptor manipulation code, while the derived descriptor classes merely add construction and assignment code specific to their type
- Identify the correct and incorrect use of modifier methods in the `TDesC` and `TDes` classes
- Recognize that there is no `HBuf` class, but that `RBuf` can be used instead as a modifiable dynamically allocated descriptor

7.3 The Inheritance Hierarchy of the Descriptor Classes

- Know the inheritance hierarchy of the descriptor classes
- Understand the memory efficiency of the descriptor class inheritance model and its implications

7.4 Using the Descriptor APIs

- Understand that the descriptor base classes `TDesC` and `TDes` cannot be instantiated

- Understand the difference between `Size()`, `Length()` and `MaxLength()` descriptor methods
- Understand the difference between `Copy()` and `Set()` descriptor methods and how to use assignment correctly

7.5 Descriptors as Function Parameters

- Understand that the correct way to specify a descriptor as a function parameter is to use a reference, for both constant data and data that may be modified by the function in question.

7.6 Correct Use of the Dynamic Descriptor Classes

- Identify the correct techniques and methods to instantiate an `HBuFC` heap buffer object
- Recognize and demonstrate knowledge of how to use the new descriptor class `RBuF`

7.7 Common Inefficiencies in Descriptor Usage

- Know that `TFileName` objects should not be used indiscriminately, because of the stack space each consumes
- Understand when to dereference an `HBuFC` object directly, and when to call `Des()` to obtain a modifiable descriptor (`TDes&`)

7.8 Literal Descriptors

- Know how to manipulate literal descriptors and know that those specified using `_L` are deprecated
- Specify the difference between literal descriptors using `_L` and those using `_LIT` and the disadvantages of using the former

7.9 Descriptor Conversion

- Know how to convert 8-bit descriptors into 16-bit descriptors and vice versa using the descriptor `Copy()` method or the `CnvUtfConverter` class
- Recognize how to read data from file into an 8-bit descriptor and then ‘translate’ the data to 16-bit without padding, and vice versa

- Know how to use the `TLex` class to convert a descriptor to a number, and `TDes::Num()` to convert a number to a descriptor

8 Dynamic Arrays

8.1 Dynamic Arrays in Symbian OS

- Demonstrate an understanding of the basics of Symbian OS dynamic arrays (`CArrayX` and `RArray` families)
- Understand the different types of Symbian OS dynamic arrays with respect to memory arrangement (flat or segmented), object storage (within array or elsewhere), object length (fixed or variable) and object ownership.
- Recognize the appropriate circumstances for using a segmented-buffer array class rather than a flat array class

8.2 `RArray`, `RPointerArray` or `CArrayX`?

- Know the reasons for preferring `RArrayX` to `CArrayX`, and the exceptional cases where `CArrayX` classes are a better choice

8.3 Array Granularities

- Understand the meaning of array granularity and capacity
- Know how to choose the granularity of an array as appropriate to its intended use

8.4 Array Sorting and Searching

- Demonstrate an understanding of how to sort and seek in dynamic arrays
- Recognize that `RArray`, `RPointerArray` and the `CArrayX` family can all be sorted, although the `CArrayX` classes are not as efficient

8.5 `TFixedArray`

- Recognize that, when a dynamic array is not required, the `TFixedArray` class should be preferred over a C++ array, since it gives the benefit of bounds checking (debug-only or debug and release)

9 Active Objects

9.1 Event-Driven Multitasking on Symbian OS

- Demonstrate an understanding of the difference between synchronous and asynchronous requests and be able to differentiate between typical examples of each
- Recognize the typical use of active objects to allow asynchronous tasks to be requested without blocking a thread
- Understand the difference between multitasking using multiple threads and multiple active objects, and why the latter is preferred in Symbian OS code

9.2 Class `CActive`

- Understand the significance of an active object's priority level
- Recognize that the active object event handler method (`RunL()`) is non-pre-emptive
- Know the inheritance characteristics of active objects, and the functions they are required to implement and override
- Know how to correctly construct, use and destroy an active object

9.3 The Active Scheduler

- Understand the role and characteristics of the active scheduler
- Know that `CActiveScheduler::Start()` should only be called after at least one active object has an outstanding request
- Recognize that a typical reason for a thread to fail to handle events may be that the active scheduler has not been started or has been stopped prematurely
- Understand that `CActiveScheduler` may be sub-classed, and the reasons for creating a derived active scheduler class

9.4 Canceling an Outstanding Request

- Understand the different paths in code that the active object uses when an asynchronous request completes normally, and as the result of a call to `Cancel()`

9.5 Background Tasks

- Understand how to use an active object to carry out a long-running (or background) task
- Demonstrate an understanding of how self-completion is implemented

9.6 Common Problems

- Know some of the possible causes of stray signal panics, unresponsive event handling and blocked threads

10 System Structure

10.1 DLLs in Symbian OS

- Know and understand the characteristics of polymorphic interface and shared library (static) DLLs
- Know that UID2 values are used to distinguish between static and polymorphic DLLs, and between plug-in types
- For a shared library, understand which functions must be exported if other binary components are to be able to access them
- Know that Symbian OS does not allow library lookup by name but only by ordinal

10.2 Writable Static Data

- Recognize that writable static data is not allowed in DLLs on EKA1 and discouraged on EKA2
- Know the basic porting strategies for removing writable static data from DLLs

10.3 Executables in ROM and RAM

- Recognize the correctness of basic statements about Symbian OS execution of DLLs and EXEs in ROM and RAM

10.4 Threads and Processes

- Recognize the correctness of basic statements about threads and processes on Symbian OS

- Recognize the role and the characteristics of the synchronization primitives `RMutex`, `RCriticalSection` and `RSemaphore`

10.5 Inter-Process Communication (IPC)

- Recognize the preferred mechanisms for IPC on Symbian OS (client–server, publish and subscribe and message queues), and demonstrate awareness of which mechanism is most appropriate for given scenarios
- Understand the use of publish and subscribe to retrieve and subscribe to changes in system-wide properties, including the role of platform security in protecting properties against malicious manipulation

10.6 Recognizers

- Recognize correct statements about the role of recognizers in Symbian OS

10.7 Panics and Assertions

- Know the type of parameters to pass to `User::Panic()` and understand how to make them meaningful
- Understand the use of `__ASSERT_DEBUG` statements to detect programming errors in debug code by breaking the flow of code execution using a panic
- Recognize that `__ASSERT_ALWAYS` should be used more sparingly because it will test statements in released code too and cause code to panic if the assertion fails

11 Client–Server Framework

11.1 The Client–Server Pattern

- Know the structure and benefits of the client–server framework
- Understand the different roles of system and transient servers, and match the appropriate server type to examples of server applications

11.2 Fundamentals of the Symbian OS Client–Server Framework

- Know the fundamentals of the Symbian OS client–server implementation

11.3 Symbian OS Client–Server Classes

- Know the classes used by the Symbian OS client–server framework, and basic information about the role of each
- Recognize the objects that a server must instantiate when it starts up
- Understand the mechanism used to prevent the spoofing of servers in Symbian OS

11.4 Client–Server Data Transfer

- Know the basics of how clients and servers transfer data for synchronous and asynchronous requests
- Recognize the correct code to transfer data from a client derived from `RSessionBase` to a Symbian OS server
- Know how to submit both synchronous and asynchronous client–server requests
- Know how to convert basic and custom data types into the appropriate payload which can be passed to the server, as both read-only and read/write request arguments

11.5 Impact of the Client–Server Framework

- Understand the potential impact on run-time speed from using a client–server session and differentiate between circumstances where it is useful or necessary and where it is inefficient
- Recognize scenarios where an implementation which uses client subsessions with the server would be recommended
- Understand the impact of the context switch required when making a client–server request, and the best way to manage communication between a client and a server to maximize run-time efficiency

12 File Server and Streams

12.1 The Symbian OS File System

- Understand the role of the file server in the system
- Know the basic functionality offered by class `RFS`
- Recognize code which correctly opens a filesaver session (`RFS`) and a file subsession (`RFile`) and reads from and writes to the file

- Know the characteristics of the four `RFile` API methods which open a file
- Understand how `TParse` can be used to manipulate and query file names

12.2 Streams and Stores

- Know the reasons why use of the stream APIs may be preferred over use of `RFile`
- Understand how to use the stream and store classes to manage large documents most efficiently
- Be able to recognize the Symbian OS store and stream classes and know the basic characteristics of each (for example base class, memory storage, persistence, modification, etc.)
- Understand how to use `ExternalizeL()` and operator `<<` with `RWriteStream` to write an object to a stream, and `InternalizeL()` and operator `>>` with `RReadStream` to read it back
- Recognize that operators `>>` and `<<` can leave

13 Sockets

13.1 Introducing Sockets

- Recognize correct high-level statements which define and describe a network socket
- Recognize correct statements about transport independence
- Know the difference between connected and connectionless sockets
- Differentiate between streamed and datagram communication and their relationship with connected/connectionless sockets

13.2 The Symbian OS Sockets Architecture

- Demonstrate a basic understanding of the support for sockets on Symbian OS
- Recognize the characteristics of the `RSocketServ`, `RSocket` and `RHostResolver` classes
- Understand the role and purpose of PRT protocol modules

13.3 Using Symbian OS Sockets

- Recognize correct patterns for opening and configuring connected and connectionless sockets
- Know which `RSocket` API methods should be used for connected and unconnected sockets to send and receive data
- Know the characteristics of the synchronous and asynchronous methods for closing an `RSocket` subsession

14 Tool Chain

14.1 Build Tools

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

14.2 Hardware Builds

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

14.3 Installing an Application to Phone Hardware

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

14.4 The Symbian OS Emulator

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

15 Platform Security

15.1 The Trust Model

- Understand what is meant by the axiom “a process is a unit of trust” and how Symbian OS enforces this
- Understand the purpose of the Trusted Computing Base and why it is important
- Recognize that a number of Symbian OS APIs do not require security checks before they can be used
- Know that self-signed software that does not use sensitive system services is “untrusted” and can be installed and run on the phone, although it is effectively “sandboxed”

15.2 Capability Model

- Understand the relationship between capabilities and the Trusted Computing Base (TCB)
- Understand the concept of user capabilities and their relationship to the Trusted Computing Environment (TCE)
- Understand the relationship between the TCB/TCE, capability assignment, software install as the “gatekeeper” and the role of application signing
- Recognize the different groups of capabilities, demonstrating a broad understanding of the privileges granted
- Recognize how to specify platform security capabilities within an MMP file
- Demonstrate an understanding of the capability rules

15.3 Data Caging

- Understand how data caging works to protect all types of files via the three special directories (\sys, \resource and \private); in particular, that data caging is used to partition all executables in the file system so, once trusted, they are protected from modification
- Understand the implications of data caging for naming executable code
- Recognize that data caging can be used to provide a secure area for an application’s data

- Recognize the capabilities needed to read from and write to specific directories and subdirectories
- Know that DLLs do not have a private data-caged area and use that of the process in which they are loaded, and that this directory can be acquired by the DLL using the file system methods `RFs::PrivatePath()`

15.4 Secure Identifier, Vendor Identifier and Unique Identifier

- Explain what a Secure Identifier (SID) is, where it is defined and what it is used for
- Understand the similarities and differences between a Secure Identifier (SID), a Vendor Identifier (VID) and a binary's Unique Identifiers (UID)
- Know the rules by which an application is identified, according to the specification of SID, VID and UID
- Understand that SID and VID may be assigned, but are not relevant, to DLLs
- Recognize how to specify VID and SID within an MMP file
- Understand that UIDs are now split into 2 groups (protected and unprotected ranges) with different implications for test and commercial code

15.5 Application Design for a Secure Platform

- Demonstrate an understanding of the key considerations when writing a secure application, including the parties interested in application security, typical attacks, countermeasures and secure application design, and the costs of various countermeasures

15.6 Releasing a Secure Application on Symbian OS v9

- Understand the basic process of testing and releasing a signed V9 application

15.7 The Native Software Installer

- Recognize the key functions of the v9 Native Software Installer, including the compatibility break in SIS file format between v9 and previous versions of Symbian OS

16 Compatibility

16.1 Levels of Compatibility

- Demonstrate an understanding of source, binary, library, semantic and forward/backward compatibility

16.2 Preventing Compatibility Breaks – What Cannot Be Changed?

- Recognize which attributes of a class are necessary for a change in the size of the class data not to break compatibility
- Understand which class-level changes will break source compatibility
- Understand which class-level changes will break binary compatibility
- Understand which library-level changes will break binary compatibility
- Understand which function-level changes will break binary and source compatibility
- Differentiate between derivable and non-derivable C++ classes in terms of what cannot be changed without breaking binary compatibility

16.3 What Can Be Changed Without Breaking Compatibility?

- Understand which class-level changes will not break source compatibility
- Understand which class-level changes will not break binary compatibility
- Understand which library-level changes will not break binary compatibility
- Understand which function-level changes will not break binary and source compatibility
- Differentiate between derivable and non-derivable C++ classes in terms of what can be changed without breaking binary compatibility

16.4 Best Practice – Designing to Ensure Future Compatibility

- Recognize best practice for maintaining source and binary compatibility
- Recognize the coupling arising from the use of inline functions and differentiate between cases where it will make maintaining binary compatibility more difficult and where it will be less significant

