

Symbian OS Application Engine

Details

Level:	Intermediate
Duration:	4 days
Development language:	C++
Experience:	Delegates should have already attended the Symbian OS Essentials course or have equivalent experience. They should be proficient in C++ and have a good understanding of OO techniques. Further a basic understanding of platform security is required. Familiarity with Carbide.c++ IDE is also beneficial.



Target audience

This course is for application developers focusing on the implementation of engines and other application support services and will give developers an understanding of engine design issues and experience of engine implementation issues.

Description

The course presents various design techniques and system frameworks of interest to engine developers as well as some of the key Symbian OS APIs used in engines. Areas covered in the course include DLLs, Active Objects, Client/Server, Sockets, DBMS and binary compatibility. Materials about DBMS are provided as a self study guide.

This is a hands-on course consisting of approximately 50% lectures and 50% practical work. The course trains generic Symbian OS programming methodologies, all of which are applicable to all phone types, all emulators, all UI systems and all versions of the OS.

Applications for Symbian OS are typically designed as two separate modules, a user interface and an engine. The engine implements all the functionality of the application, without any dependence on a particular user interface or look-and-feel. Symbian OS provides numerous component libraries and APIs to allow developers to write sophisticated engine functionality. This course provides a thorough grounding in these APIs, together with various design techniques and system frameworks of interest to engine developers.

Objectives

Upon completing this course, participants will be able to:

- Describe the UI-engine and MVC models for Symbian OS applications
- Understand the design issues involved in developing Symbian OS engines
- Build application engines as separate DLLs that pay attention to binary compatibility issues
- Recognize the need for servers and make use of the Client/Server and active object frameworks
- Implement key services that utilize the Socket server.

Tools and platform

During the course we will use the Nokia Carbide.c++ OEM v2 IDE environment. Carbide.c++ can be downloaded from Forum Nokia free of charge.

Exercises and course materials are adapted for the Series 60 5th Edition SDK emulator.

Agenda

Day 1

Course overview

- Overview of the course and high-level application design issues
- UI-engine and MVC application models

This module gives guidance on how to split a Symbian OS application into a user interface, application controller, application engine and service providers by design (the MVC paradigm). Considerations are given on when to implement a service provider as a server. Binary compatibility and platform security issues are also addressed briefly.

Building and testing

- Writing DLLs
- Using RTest and test harnesses

This module introduces the different types of DLLs within Symbian OS: shared library DLLs, provider libraries and ECOM plugins. Capability rules are given for loading DLLs. In an exercise a shared library DLL is created. The module describes how to use the RTest class and Test Driver to test the shared library DLLs.

Day 2

Binary Compatibility(BC)

- Maintaining BC
- When to break it!

The module explains the concept of binary compatibility (source, binary, backward and forward compatibility). It further explains how BC can be broken (header file, import library, exports table, behaviour). The module gives practical advice on how to prevent BC to be broken. For this we look at access specifiers, order of data members, inline methods, class size, enumerations, function ordinals, virtual methods, changing data types, const-ness and behaviour. To make your code future proof freeze files (.def) are introduced as well as showing techniques like hiding the vtable, making classes non-derivable and declaring reserved methods and data members. Also careful consideration should be given to which function are exported and published. The module ends with a quiz exercise to validate the understanding of binary compatibility by the candidate.

Active Objects

- Using and writing Active Objects

This module starts with an overview of active objects as introduced in the Symbian OS Essentials course. After that a walkthrough is given on how to write the source code for an active object class. Active objects can issue a sequence of requests and may need to keep track of their current state. If the number of states grows the state design pattern can be applied (one object per state and a single manager object). Active objects can also be used for long-running background tasks. Finally trade off considerations are given on when to use active objects, multi-threading or writing a new server.

Day 3

Client-server

- When and how to write a server
- Client-side and server-side issues

The client-server module starts by providing considerations which should be taken into account on deciding whether to write a new server or to take another approach. Then design issues (key classes, messages) are covered as well as performance costs, data exchange issues, multi-threaded servers, startup and shutdown and platform security issues. The support for subsessions and the implementation of the client side DLL with the service API is explained. The startup of the server and data exchange between client and server using package buffers is covered. On the server side the CServer2 class to handle startup and the CSessi on2 class to handle messages and data exchange need to be implemented. Secure servers are derived from CPol i cyServer and implement a TPol i cy security policy possibly with customised checks for parameters of specific messages.

Day 4

Sockets and TCP/IP

- Introduction to the Socket server and its key classes
- Implementing Socket-based communication
- Architecture for Socket-based communications

The module introduces the socket server basics (Symbian OS communication components, protocol modules) and key classes (RSocketServ, RSocket, RHostResolver) as well as using streams and datagrams. The lifecycle for connected (create, bind, connect, listen, accept, close) and connectionless sockets is shown. Active objects are used to encapsulated asynchronous requests on sockets to connect, send and receive data. A coding exercise brings the sockets alive in a simple chat application on the emulator.

Self Study

DBMS

- Symbian OS DBMS server
- Key construction and manipulating databases using C++ and SQL APIs

This final module is left to the candidate as a self study module. It introduces the multi-user database server as embedded in Symbian OS. The DBMS server allows the creation of physical database files which contains multiple tables and indexes in a single file. Tables and indexes are created either through an C++ API or by using SQL. Records in the tables can be queried, inserted and modified using SQL queries. Through the C++ API transactions and change notifications are also supported. Dynamic arrays are suitable classes to store the result sets of queries.