



# Contents

- 1 Introduction
- 2 Overview of iPhone OS and Symbian OS
- 3 Languages Used for Native Development
- 4 Getting Started
  - ◆ 4.1 IDEs
  - ◆ 4.2 Code: Similarities and differences
    - ◇ 4.2.1 Similarities
    - ◇ 4.2.2 Differences
  - ◆ 4.3 Key Points to Remember
    - ◇ 4.3.1 Methods
    - ◇ 4.3.2 Methods With Arguments
    - ◇ 4.3.3 Nested Messages
    - ◇ 4.3.4 Allocation Of Object
    - ◇ 4.3.5 Deallocation Of Object
- 5 Porting Application to Symbian
  - ◆ 5.1 Main Function
  - ◆ 5.2 Porting GUI Application to Symbian
    - ◇ 5.2.1 GUI Overview
    - ◇ 5.2.2 Orientation In UI Application
  - ◆ 5.3 Graphics And Drawing
  - ◆ 5.4 Event Handling
  - ◆ 5.5 DataBase Handling
  - ◆ 5.6 Communication Handling
  - ◆ 5.7 Sensor Handling
  - ◆ 5.8 MultiMedia Framework
  - ◆ 5.9 XML Handling
  - ◆ 5.10 Security Framework
  - ◆ 5.11 Application Signing Overview
  - ◆ 5.12 Application Bundle
- 6 Technical References
  - ◆ 6.1 External
  - ◆ 6.2 Internal
    - ◇ 6.2.1 Introductory References

- ◇ [6.2.2 Signing And Security References](#)
- ◇ [6.2.3 Binary Compatibility](#)
- ◇ [6.2.4 Tools](#)
- ◆ [6.3 Books](#)
- ◆ [6.4 WebSites](#)

## Introduction

When someone familiar with developing for iPhone (Mac OS) OS moves to developing for Symbian OS, there is a period of uncertainty about where to start. The APIs, tools and development environments are so dissimilar that it comes as quite a shock to many.

This article is meant to provide beginning and intermediate iPhone OS developers with an introduction to porting applications from iPhone to Symbian OS v9.x.



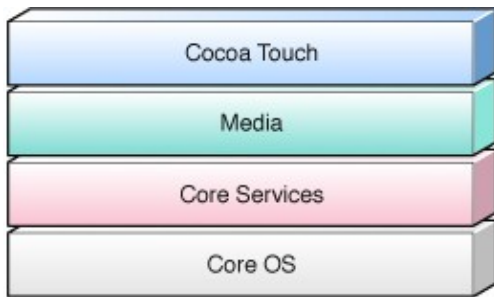
The source porting should be rather easy, considering the similarities in functions provided by the languages, and the biggest differences in the code will appear in the GUI section.

iPhone OS development, as most of Mac development, is usually extensively based on MVC pattern. Thanks to this, most of the porting work should concern the graphical interface, while the model and controller part should remain close to basic translation, following the rules described below.

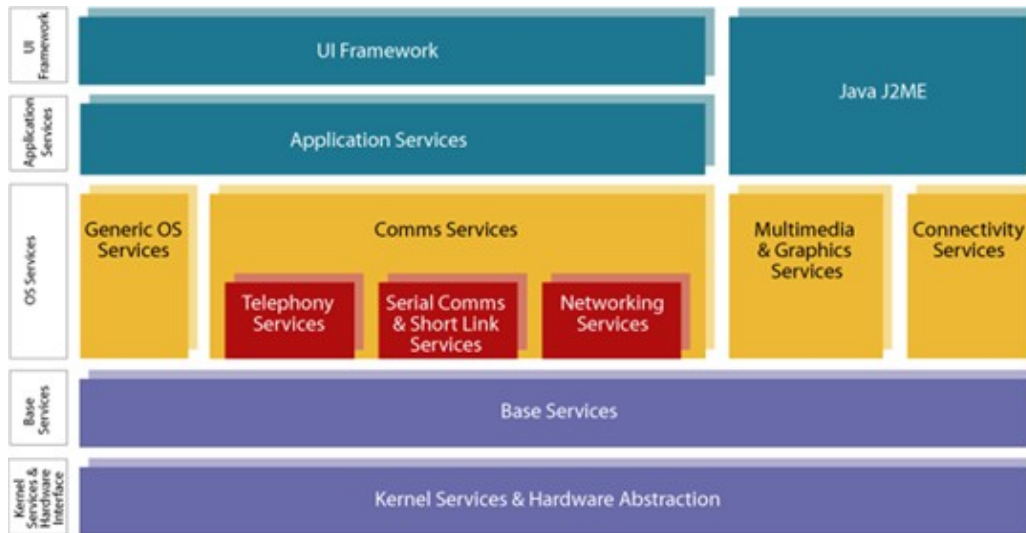
Of course, the more specific iPhone functions used, the more complex the port (GPS use, accelerator integration, SQLite data storage ...).

# Overview of iPhone OS and Symbian OS

## iPhone OS Architecture Overview



## Symbian OS Architecture Overview



## Languages Used for Native Development

iPhone OS: Objective-C

Symbian OS: Symbian C++

## Getting Started

### IDEs

- Symbian OS: [Carbide.c++](#)
- iPhone OS: [XCode](#)

## Code: Similarities and differences

Let's take a look at some code. The following table compares a code with the same logic written in Objective-C and Symbian/C++. The first thing to notice is that it is necessary to write some more lines of code in Symbian/C++ if compared to Objective-C. This is because some coding conventions that are used in Symbian/C++ and that characterize the way of programming. It is worth mention that Symbian/C++ is C++ with some coding convention that are related to memory management, naming of elements (classes, variables ...), etc. In the next lines the similarities among the languages will be highlighted.

*NOTE: If you are not familiarized with C++ this can be a start point, but it is really recommended more information about the topic <<insert useful link here>>. The comparisons will be among Objective-C with C++ and Symbian/C++ (that involves the coding conventions)*

### Similarities

- The basic similarity is that both languages Objective-C and C++ came from the necessity to increment the C language with support to object oriented skills.
- Other aspect is that both languages uses the same extensions for the files designed to have the classes declarations: **.h**. Regarding the source file, Objective-C uses the extension **.mm** and C++ uses the extension **.cpp**
- In both languages there is a clear separation between classes declarations and implementations. The separation is clear on the code where it can be seen '@interface' keyword of Objective-C matching the 'class' keyword of C++. In fact, except the code added for object construction of Symbian/C++ every line of code of the Brad Cox's language (creator of Objective-C) has a very similar line of code in C++.
- The scope delimiter of the instance variables are also the same, i.e., @private, @protected and @public delimiters are available in C++.
- Other similarity is that in Symbian/C++ there is also a common base class 'CBase', similar to 'NSObject', that is used in all classes of type 'C'. In Symbian/C++ every class has a type. The 'C' classes are dynamic allocated on heap and have a special construction mechanism called Two-phase construction. Besides this type are others like 'T' classes that are usually used for stack based objects. More information regarding the types of classes of Symbian/C++ can be found here
  - ◆ C class
  - ◆ R class
  - ◆ M class
  - ◆ T class

### Differences

- There is no standard way to synthesize methods in C++. So every @property on Obj-C code has to be implemented. In this case the property myProperty has the equivalent methods **TInt myProperty()** and **void setMyProperty(TInt property)**.
- Static methods are declared with a **static** keyword in C++ instead of + sign in Obj-C. Regarding the variables the same keyword is used in both cases.

## Porting\_iPhone\_native\_(Objective-C)\_applications\_to\_S60\_5th\_Edition

- You need to write special code for object construction in classes of type *C*. This special code is called two phase construction. The first construction step consist of calling the C++ constructor that shouldn't alloc any resource. In the second phase, you call the method ConstructL that effectively alloc the resources for the object. Also, in the wo phase construction you should be aware of the cleanup stack mechanism that Symbian adopts and use it accordingly (see Cleanupstack).



```
#import <NSObject.h>

@class Bar;
@interface Foo : NSObject
{
@private
    double x;
    Bar* mybar;
    int myProperty;
}

@property int myProperty;

-(void) useBar;

-(int) doSomething:(int) arg;

-(int) doOtherThing:(int) arg1 :(int) arg2;

+(double) doThingsForThisClass:(double) arg;

@end

@implementation Foo

@synthesize myProperty;

-(void) useBar
{
}

-(int) doSomething:(int) arg
{
    return arg;
}

-(int) doOtherThing:(int) arg1 :(int) arg2
{
    return arg1+arg2;
}
```

```
#ifndef __FOO_H__
#define __FOO_H__

#include <e32def.h>

class CBar;

class CFoo : public CBase
{
private:
    TReal iX;
    CBar* iMyBar;
    TInt iMyProperty;

    /* Code added for object construction
    in Symbian/C++*/
    CFoo(){};
    void ConstructL(){};

public:
    /* Code added for object construction
    in Symbian/C++*/
    static CFoo * NewL();

    TInt myProperty();
    void setMyProperty(TInt property);

    void useBar();

    TInt doSomething(TInt arg);

    TInt doOtherThing(TInt arg1, TInt arg2);

    static TReal doThingsForThisClass(
        TReal arg);
};

/* Code added for object construction
in Symbian/C++*/
CFoo* CFoo::NewL()
{
    CFoo* instance = new (ELeave) CFoo;
```

## Porting\_iPhone\_native\_(Objective-C)\_applications\_to\_S60\_5th\_Edition

```
+ (double) doThingsForThisClass:(double) arg
{
    return arg;
}
@end

. . .
instance->ConstructL();
. . .
return instance;
}

TInt Foo::myProperty()
{
    return myProperty;
}

void Foo::setMyProperty(TInt property)
{
    iMyProperty = property;
}

void Foo::useBar()
{
}

TInt Foo::doSomething(TInt arg)
{
    return arg;
}

TInt Foo::doOtherThing(TInt arg1, TInt arg2)
{
    return arg1+arg2;
}

TReal Foo::doThingsForThisClass(TReal arg)
{
    return arg;
}

#endif
```

### Key Points to Remember

- Use #include instead of #import in Symbian C++
- Use NULL instead of nil in Symbian C++
- It is really a good practice to separate the UI and Engine part of the code separately following the MVC Design Patterns in Symbian.

### Methods



### Objective-C Messages

#### Objective-C Example:

```
/* Objective-C is a language with
   ?function calls using brackets?.
   [object doSomething]; ;*/

[receiver message];
```

To get an object to do something, programmer has to send it a message telling it to apply a method.

Note: The receiver is an object, and the message tells it what to do.



#### Symbian C++ Example:

```
/* Symbian C++ is a language with
   ?function calls using parentheses?.
   object.doSomething();*/

Receiver* receiver = new(ELeave) Receiver();
receiver->Message();

// deletion of the allocated object
delete receiver;
receiver = NULL;
```

### Methods With Arguments



#### Syntax:

```
[receiver message:Argument1];
```

#### Objective-C Example:

```
// example 1
[myRectangle setWidth:20.0];

// example 2
[myRectangle setOriginX: 30.0 Y: 50.0];
```



#### Symbian C++ Example:

```
TRect rect(0,0,0,0);

rect.SetRect(10,20,30,40);
```

### Nested Messages



**Objective-C Example:**

```
[myRectangle setPrimaryColor:[otherRect primaryColor]];
```

One message expression can be nested inside another.



**Symbian C++ Example:**

```
TRect rectA(10,20,30,40);  
rectA.SetSize(TSize(30,40));
```

**Allocation Of Object**



**Objective-C Example:**

```
Object *object = [[Object alloc] init];
```

There is also an autorelease pool that can be used to avoid manual deallocation, but it must be used with caution, since deallocation occurs at the end of the context (usually as the method returns), while you may need to deallocate it sooner for many reasons.



**Symbian C++ Example:**

```
Object *object = new(Eleave) Object();
```

**Deallocation Of Object**



**Objective-C Example:**

```
// deallocation of objects  
[object release] ;  
[super dealloc];
```



**Symbian C++ Example:**

```
delete object;  
object = NULL;
```

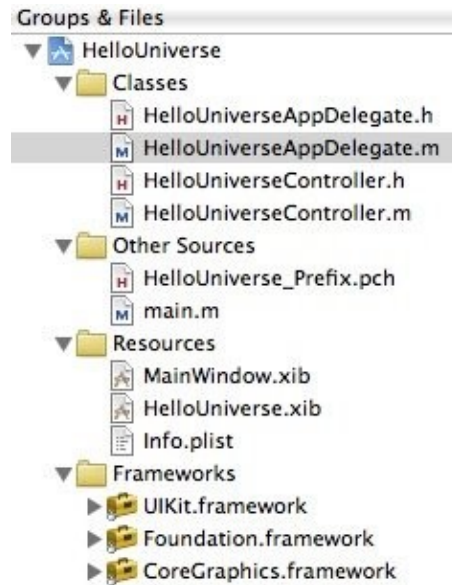
In second line we are sending a message to a superclass NSObject in most of the case to do its cleanup. If we dont do this the object will not remove and it is a memory leak.

## Porting Application to Symbian



- [How to create project in Carbide.c++ for the s60](#)
- [Directory layout of Symbian projects](#)

List of files look like in Xcode



- [RSG File](#)

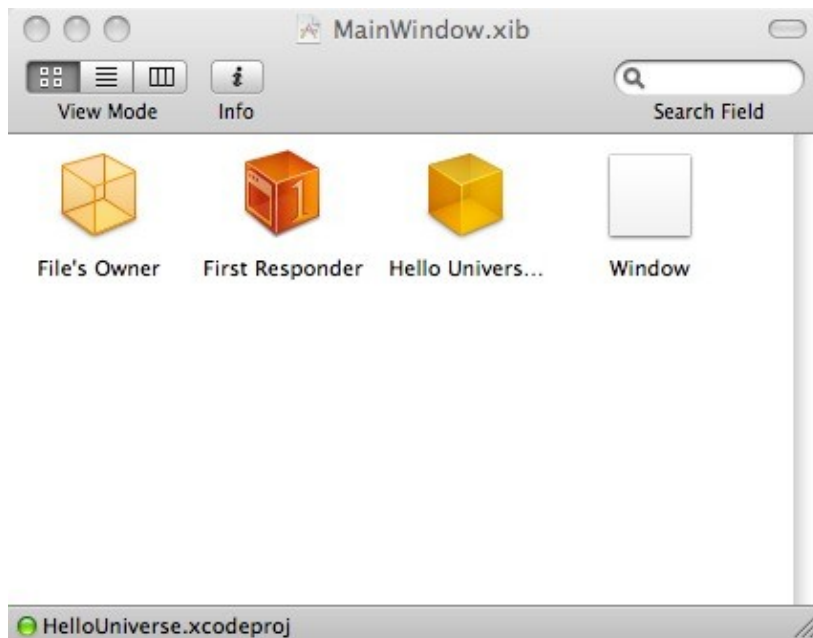
Interface Builder iPhone:

Every project gets one nib file by called which is called "MainWindow.xib" which can be found under "Resources". An iPhone application has only one window (MainWindow.xib). Double click on "MainWindow.xib" to launch the Interface Builder, which will open four windows and one of the window will look like this.

Every nib file has atleast two files. File's Owner and First Responder which cannot be deleted. Every other objects apart from the first two,

represents an instance of an object which gets created when the nib file loads.

- File's Owner simply shows that it owns the object in the nib file.
- First Responder tells us which object are we currently interacting with; like the textbox, buttons...
- The third object which is special to the MainWindow.xib file is called "Hello Universe App Delegate" and this file represents "HelloUniverseAppDelegate" class. Last but not the least the view represents the object which we design in our apps.



## Main Function



```
#import <UIKit/UIKit.h>
int main(int argc, char *argv[])
```



```
#include <eikstart.h>
#include "CUSSDApplication.h"
```

## Porting\_iPhone\_native\_(Objective-C)\_applications\_to\_S60\_5th\_Edition

```
{
    NSAutoreleasePool * pool =
        [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(
        argc, argv, nil, nil);
    [pool release];
    return retVal;
}

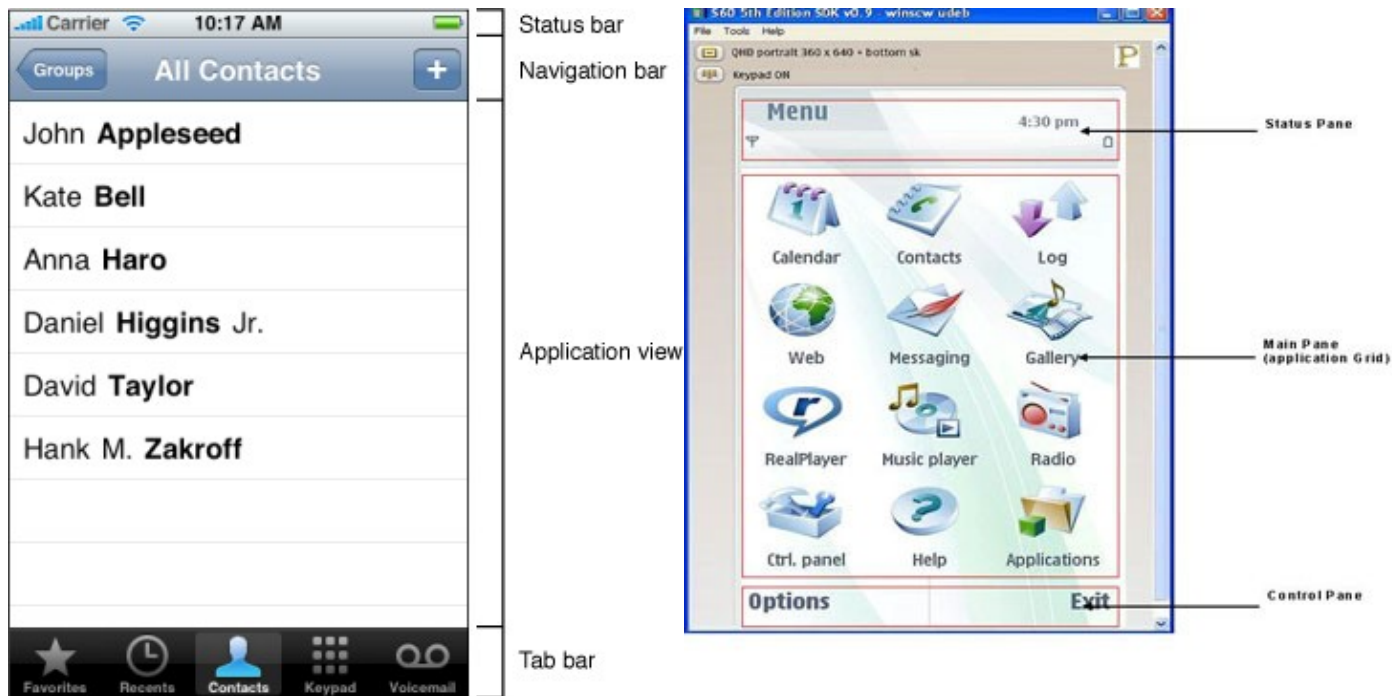
LOCAL_C CAppApplication* NewApplication()
{
    return new CMyApplication;
}
/* Entry Point for Symbian GUI */

GLDEF_C TInt E32Main()
{
    return EikStart::RunApplication(
        NewApplication
    );
}
```

- GUI vs. Console
- Polymorphic DLL
- Static dll

## Porting GUI Application to Symbian

### GUI Overview



- GUI Framework

## Orientation In UI Application



- Change screen orientation of UI application

The UIInterfaceOrientation key provides a hint to iPhone OS.

## Graphics And Drawing



- S60 has the support of OpenGL ES.

- iPhone OS supports drawing through the OpenGL ES (OpenGL ES v1.1), Quartz, UIKit, or Core Animation framework. The UIView class makes the update process easier and more efficient.

## Supported Images

Format	Extension	iPhone	S60
Portable Network Graphic(PNG)	.png	yes	yes
Tagged Image File Format(TIFF)	.tiff, .tif	yes	
Joint Photographic Experts Group(JPEG)	.jpeg, .jpg	yes	yes
Graphic Interchange Format(GIF)	.gif	yes	yes
Windows Bitmap Format (DIB)	.bmp, .BMPf	yes	yes
Windows Icon Format	.ico	yes	
Windows Cursor	.cur	yes	
XWindow bitmap	.xbm	yes	
MBM	.mbm	no	yes

### Simple Graphics API

iPhone	Use	S60
UIImage	class for displaying images	CFbsBitmap,CGullIcon
UIColor	provides basic support for device colors	TRgb
UIFont	font information	CFont
UIScreen	provides basic information about the screen	CWsScreenDevice

### Event Handling



- Does Not Support the Key Events.



- Does not support the multi-touch till now.
- [Using basic touch gestures](#)
- [A tour to the S60 Touch UI components](#)

### DataBase Handling



- [DBMS](#)

### Communication Handling



- [Symbian C++ Messaging Articles](#)
- [Symbian C++ Networking Articles](#)
- [Symbian C++ Connectivity Articles](#)
- [Symbian C++ Telephony Articles](#)

Apple provides built-in support for the http, mailto, tel, and sms URL schemes. It also supports http?based URLs targeted at the Maps, YouTube, and iPod applications. Applications can register their own custom URL schemes as well. To communicate with an application, create an *NSURL* object with some properly formatted content and pass it to the openURL: method of the shared *UIApplication* object.

### InterProcess Communication

- [Message Queues](#)
- [Inter Process Communication in Symbian](#)

### Sensor Handling



- [Nokia Sensor APIs](#)
- [S60 Sensor Framework](#)

### MultiMedia Framework



- [Symbian C++ Multimedia Articles](#)

### XML Handling



- [How to parse XML file using CParser class](#)

### Security Framework



- [Symbian C++ Security Articles](#)

## Application Signing Overview



- [Symbian C++ Application Signing Articles](#)

## Application Bundle

Objective-C	Symbian C++	Remarks
MyApp.app	MyApp.exe	The executable file containing your application?s code.
Icon.png	Icon.mbm	Represent your application on the device home screen.
MainWindow.nib	MainWindow.rss	Typically, this nib file contains the application?s main window object.This could be compared to Symbian Resource file but to a full extent.
Info.plist	Symbian Package file.	information property list, this file is a property list defining key values for the application, such as bundle ID, version number, and display name
en.lproj fr.lproj es.lproj	en.l01 fr.l01 es.l01	Localized resources.
MyApp.app	MyApp.SISX	Final ouput file .

## Technical References

### External

- [Apple](#)
- [Writing portable code and maintaining ports](#)

### Internal

#### Introductory References

- [How do I start programming for Symbian OS?](#)
- [SIS](#)
- [How to define application icon](#)
- [Design Patterns in Symbian](#)
- [MMP file](#)
- [UID](#)

#### Signing And Security References

- [Application Signing](#)
- [Advanced Package File Options](#)

#### Binary Compatibility

#### Tools

- [Symbian C++ Tools Articles Tools Articles](#)

#### Books

- [Symbian Resources and Books](#)
- [Library.forum.nokia](#)

#### WebSites

- [Forum Nokia](#)