



Code less.
Create more.
Deploy everywhere.

Qt Quick Start

This Quick Start is relevant if you want to create Qt applications for the Symbian platform. It assumes you are using the free GCCCE compiler and does not cover rebuilding Qt itself.

The first part of the tutorial, "Getting Started", shows you how to set up your *Qt for the Symbian platform v4.6.0* development environment, build an example command line application, and deploy Qt to your device.

The second part, "Getting Started with Qt using Carbide.c++", explains how you configure Carbide for Qt development, create a skeleton application using the Carbide.c++ Qt Wizard, and get it up and running on both the Symbian platform emulator and on the device.

An up-to-date version of this document can be found at the following location:

http://developer.symbian.org/wiki/index.php/Qt_Quick_Start

1. Getting Started

This section explains how you set up *Qt for the Symbian platform v4.6.0* on your Windows computer, build an example using the command line, and install the Qt binaries and demos on your Symbian mobile device.

The process has five parts:

1. Set up the Symbian C++ development environment.
2. Update the Symbian C++ Development Environment with Open C/C++ v1.6 and other patches required by Qt.
3. Install Qt, layering over the top of selected SDKs.
4. Configure the command line for Qt development.
5. Deploy Qt binaries and Open C/Open C++ to the device.

Note: At time of writing, Symbian platform development in C++ is only supported on Windows.

The developer community has provided workarounds that should allow you to develop for Symbian on Linux and Max OS X. Information is provided at the following links:

- <http://www.martin.st/symbian/> (Symbian C++)
- <http://lizardo.wordpress.com/2009/07/29/running-qt-for-s60-sdk-on-linux/> (Additional steps for Qt)

1.1. Symbian C++ Development Environment

Assuming your computer meets the standard [system requirements](#), setting up your Windows PC for Symbian platform C++ development is as simple as downloading and installing the following files (in order):

- [Perl](#) (use only version 5.6.1.638, other versions are not compatible)
- [Application Developer Toolkit \(ADT\)](#) (includes Carbide.c++ IDE)
- [Symbian Platform SDK](#) or [S60 Platform SDK \(3rd Edition FP1 or higher\)](#) (all include GCC compiler)

Windows Vista Installation and Use

There are a few [compatibility issues](#).

You will be prompted by the SDK installer to apply the *GCCE patch for MS Vista* from `<SDK>\plugins\vistapatch\` (`<SDK>` is `C:\S60\devices\S60.5th.Edition.SDK.v1.0` by default). There are instructions in the **help.txt** file in that directory.

The SDK will normally register itself in `C:\Program Files\Common Files\Symbian\devices.xml`. However, on Vista, the SDK is forced to create the file in the following location:

`%USERPROFILE%\AppData\Local\VirtualStore\Program Files\Common Files\Symbian\devices.xml`

Copy the file manually into the correct location.

1.2. Patch Symbian C++ Development Environment

1.2.1. Open C and Open C++

Qt is dependent on the [Open C](#) and [Open C++](#) compatibility layer, version 1.6.0 or higher. At time of writing, no Symbian C++ SDKs (up to at least Symbian^1) contain a version that will work with Qt. Therefore the patch must be installed over **all** SDKs you want to use for Qt development:

1. Download the latest [OpenC plugin SDK](#).
2. Unzip the installation file and run the contained **setup.exe** executable.
3. When prompted, select the target SDK over which you wish to install the plugin. Repeat steps 2 and 3 instructions for each SDK.

If you're working on a S60 3rd Edition, FP1 SDK (the earliest version on which Qt will run) you will also need to reinstall the RPipe library. This was installed by the OpenC plugin SDK to the S60_3rd_FP1_3 SDK root directory: `\Symbian\9.2\S60_3rd_FP1_3\RPipe.3.1.zip`:

1. Uncompress **RPipe.3.1.zip** over the SDK root `\Epoc32\` tree (e.g. `\Symbian\9.2\S60_3rd_FP1_3\Epoc32\`).
2. Download and replace the file `<SDK>\epoc32\tools\getexports.exe` with [getexports.exe](#).

1.2.2. Carbide Windows Compiler

If you're using the version of Carbide.c++ in ADT 1.4 you will need to update to a newer version of Carbide.C++, or apply the following patch. Extract the file into the `\x86Build` directory under your Carbide installation, e.g. `C:\Symbian\Tools\ADT.1.4\Carbide.c++\x86Build\`

- [x86Tools_3.2.5_Symbian_b482-qt.zip](#) – Windows compiler patch

If you are using any other version of Carbide, you can check the compiler version by executing **mwccsym2.exe** from the command line as shown below. The Carbide compiler needs to be at least version 3.2.5, build 482 to be able to build Qt properly.

```
C:\Symbian\Tools\ADT_1.4\Carbide.c++\x86Build\Symbian_Tools\Command_Line_Tools\mwccsym2.exe
```

1.3. Qt Development Environment

Download and install the *Qt for Open Source C++ development on Symbian* installer file below. Qt must be installed on the *same drive* as your target SDK(s), and the install path must *not* contain any spaces. When prompted, specify the S60 5th Edition SDK (Symbian^1) and any other SDKs which you wish to use with Qt.

- <http://get.qt.nokia.com/qt/source/qt-symbian-opensource-4.6.0.exe>

Note: The downloaded Qt installer is licensed under the GNU Lesser General Public License (LGPL). To use a commercial (or other) license see this page: <http://qt.nokia.com/downloads>

1.4. Configure Command Line

The ADT environment must be configured to allow command line building for the Symbian platform emulator. Using the Windows **start** button:

- **All Programs | Symbian Foundation ADT v1.4 | Carbide.c++ | Configure environment for WINSCW command line**

You can alternatively set the following environment variable:

```
SYMBIANBUILD_DEPENDENCYOFF=1
```

The Qt Command Prompt

The Qt installer provides a command prompt that has already been configured with the correct paths for Qt development; if you use the command prompt then no other setup is required. This is accessed from the Windows **start** button:

All Programs | Qt for Symbian by Nokia v4.6.0 | Qt for Symbian Command Prompt

If you want to use Qt from *any* command prompt then you will need to update the PATH environment variable to locate the Qt tools: `qmake`, `moc`, etc.

- On Windows navigate to **Control Panel | System | Advanced | Environment variables**.
- Select the **Path** variable and then insert the full path to the Qt `\bin` directory (by default this will be `C:\Qt\4.6.0\bin;`). Note that on *Windows Vista* you will need to reboot your computer for the path changes to take effect.

1.5. Build an Example Application

Open the command prompt provided in the Windows **start** button: **All Programs | Qt for Symbian by Nokia v4.6.0 | Qt for Symbian Command Prompt** and *navigate to your target project*. For this example, we will build the *animatedtiles* example code that comes with the Qt installation: `C:\Qt\4.6.0\examples\animation\animatedtiles\`.

Run *qmake* to generate the Symbian specific `bld.inf` and `.mmp` files:

```
qmake
```

Use *make debug-winscw* to build emulator debug binaries and *make run* to launch the application in the emulator:

```
make debug-winscw
make run
```

Use *make release-gcce* to build release binaries for the device. After building the binaries *make sis* is called to create an unsigned installation file. This file can be installed and run on the device after Qt has been deployed.

```
make release-gcce
make sis
```

There are also *make* targets for the RVCT compiler (*debug-armv5* and *release-armv5*). The *release-winscw* target exists, but cannot be used because release emulator binaries aren't supplied with the SDKs.

The sis file created above is self-signed. There are a number of other SIS options that can be set as environment variables or options to make – these are documented in [The Symbian platform - Introduction to Qt](#). For example, if you want to install the program immediately, make sure that the device is connected to the computer in “PC Suite” mode, and run sis target with the QT.SIS.OPTIONS=-i, like this:

```
make sis QT_SIS_OPTIONS=-i
```

1.6. Deploy Qt to a Device

The easiest way to deploy Qt to your device is to install the **qt.installer.sis** or **qt.demos.sis** installation files that were copied with Qt onto your windows computer (into the Qt “root”, by default at the location: C:\Qt\4.6.0\). These contain the Qt libraries and its dependencies (Open C and Open C++). The **qt.demos.sis** additionally contains the *fluidlauncher* demo applications – these provide an excellent example of Qt's capabilities on the mobile device.

The installation also provides **qt.sis** (Qt binaries only – depends on Open C/C++), **qt.selfsigned.sis** (a self-signed version of qt.sis), and **fluidlauncher.sis** (Depends on Qt and Open C) that may be useful in some packaging circumstances. Standalone versions of the three Open C/C++ installation files you need are located in your Symbian SDK at the following path below the EPOCROOT (the location where your SDK was installed):

- <EPOCROOT>\nokia.plugin\openc\s60opencsis\pips_s60_<version>.sis
- <EPOCROOT>\nokia.plugin\openc\s60opencsis\openc_ssl_s60_<version>.sis
- <EPOCROOT>\nokia.plugin\opencpp\s60opencppsis\stdcpp_s60_<version>.sis

Tip: At the time of writing, none of the SIS files are signed with a certificate for Samsung phones. [Samsung i8910/Omnia HD](#) owners should instead install the above Open C/Open C files and **qt.selfsigned.sis**.

1.7. Troubleshooting

The vast majority of users will have followed the above instructions without issue, and have now built an example on the command line without difficulty.

If you do encounter development environment issues, then you should:

- Test that your [development environment](#) is set up correctly.
- Follow the [Development Environment Troubleshooting Guide](#).
- Review the [Qt Known Issues](#) (on Gitorious).

2. Getting Started with Qt using Carbide.c++

If you've set up the development environment you're now ready to start creating a basic HelloWorld application using the Carbide.c++ Qt Project wizards.

This tutorial is not intended as a lesson in Qt development (although you may learn a little along the way)! It is designed to familiarise you with the Carbide.c++ IDE, and how you get your application onto the phone. If the project builds cleanly and installs then this is the best verification possible that your development environment is set up properly.

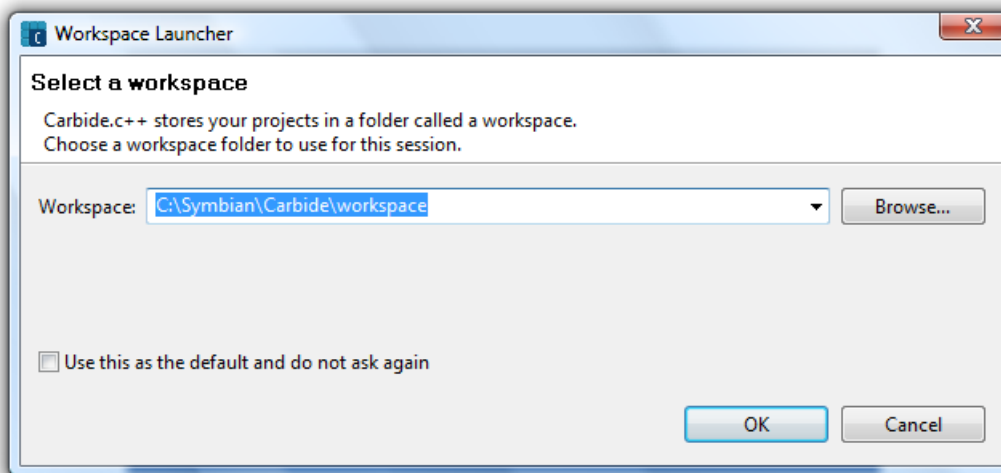


Figure 1: Carbide: Workspace Launcher

2.1. Starting Carbide.c++

The Carbide.c++ IDE was installed as part of the free [Application Development Toolkit](#). It is the only supported/official IDE for C++ development on the Symbian platform.

Carbide.c++ is launched from the Windows **start** button:

- **All Programs | Symbian Foundation ADT v<ADTVersion> | Carbide.c++ | Carbide.c++ v<CarbideVersion>**

On start, you will be prompted to select a workspace directory. The workspace directory contains any projects you've already created in the workspace and their common settings – such as code-formatting options (you can define multiple workspaces in order to separate completely different tasks). If this is the first time you've run Carbide.c++ the workspace will be empty.

If you installed the SDK to the c:\ drive, an example of a correct workspace path would be C:\Symbian\development\.

Note: Your Symbian workspace must be on the drive where you installed your SDK. You must also ensure that the path name of the workspace does not contain non-alphanumeric characters or spaces. This is because the Symbian toolchain uses command line tools that cannot read special path names.

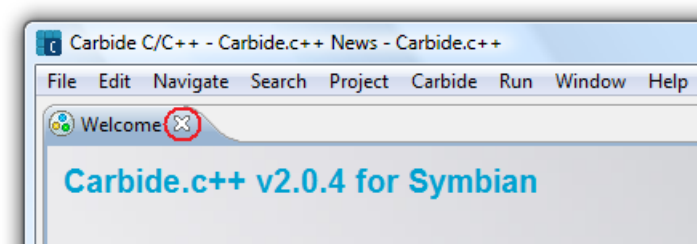


Figure 2: Carbide: Welcome Tab

Once Carbide.c++ has started, close the *Welcome* tab (by clicking the cross shown circled in red above) to see the default workspace.

2.2. Configuring Carbide.c++ for Qt Development

Carbide.c++ is “Qt aware”, in that it understands how to integrate with the Qt build system and *Qt Designer* tool. However you still need to tell it where Qt is installed. If you have multiple versions of Qt, you may also need to tell Carbide.c++ which one to use to build a particular project.

Qt versions are managed through the Carbide.c++ Qt Preferences (select Carbide menu: **Window | Preferences | Qt**).

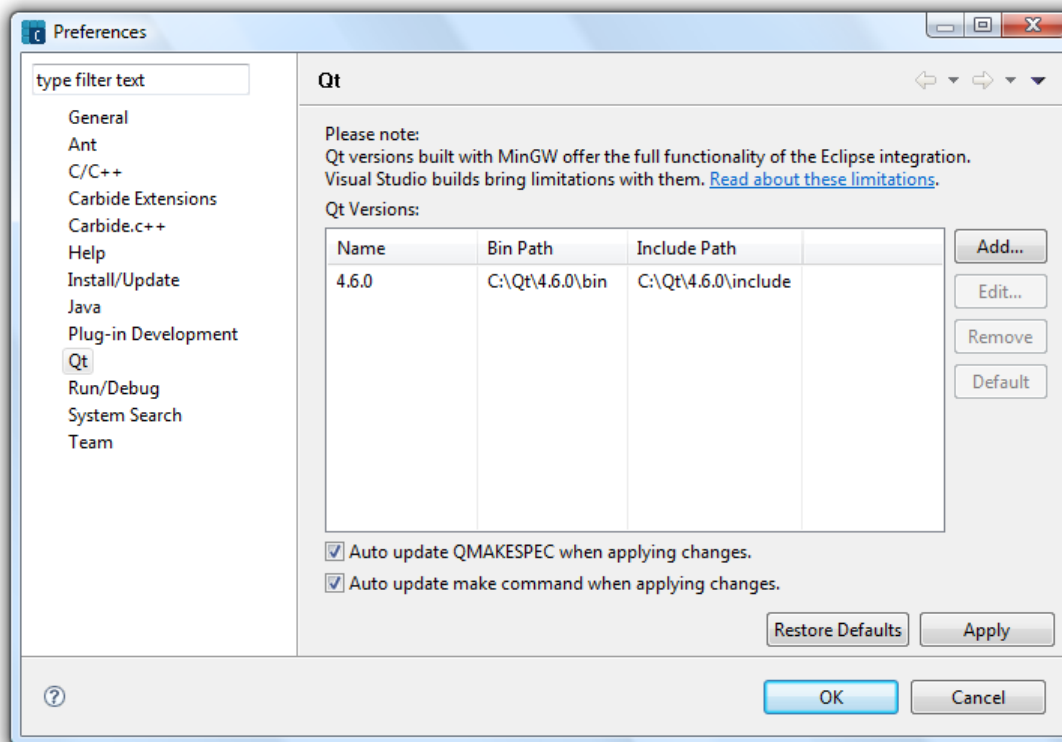


Figure 3: Carbide Preferences for Qt

If you have a number of Qt SDKs you can click **Default** to identify the selected Qt SDK as the default SDK to use for Qt projects. As a general rule you should leave the “Auto update QMAKESPEC...” and

“Auto update make command...” checkboxes selected to ensure that changes to the versions automatically update the way your projects are built.

To add a new Qt version you select **Add** to launch the *Add Qt Version* dialog.

- Enter the name of the Qt release.
- Enter the full path to the Qt \bin directory.
- Enter the full path of the Qt \include directory (this will be seeded with the correct value based on your specified \bin).
- Press **Finish** to save the dialog.

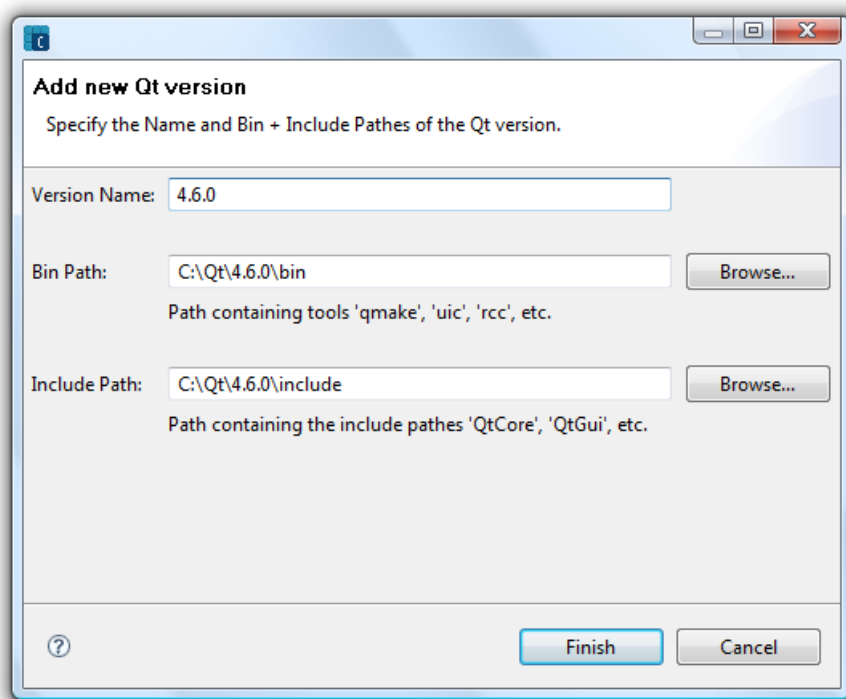


Figure 4: Carbide “Add Qt Version” Dialog

If necessary, you can change the version on a per-project basis through the Project properties settings. Select the project in project explorer and either do **File | Properties** or **right-click | Properties**.

In most cases there is no need to change the per-project settings. As you can see above these use the global default preferred Qt version, and specify for *qmake* to be run if the .pro file is changed (which is usually desirable).

Usually that is all the configuration that is required. Occasionally Carbide will not detect SDKs, or will not properly register COM plugins, resulting in the Qt Designer views not loading into Carbide; workarounds to these issues are discussed in the Troubleshooting section.

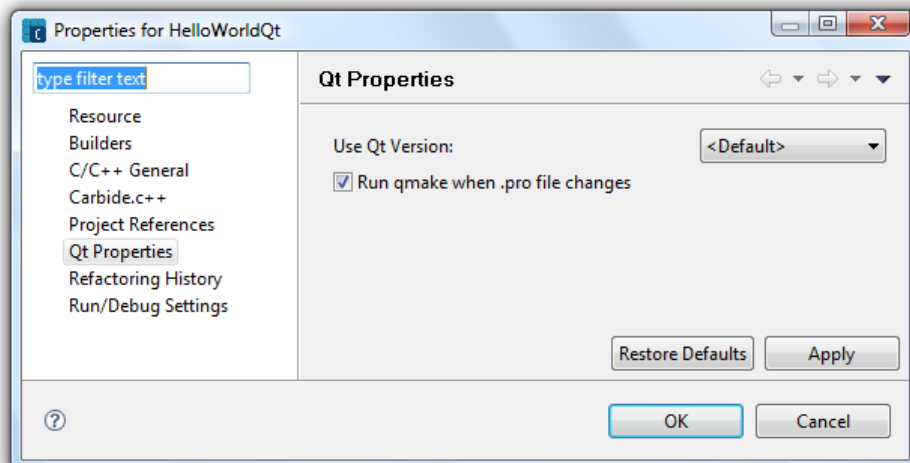


Figure 5: Carbide Project properties for Qt

2.3. Creating a Project

To launch the Carbide.c++ *Create New Project Wizard* select: **File | New | Qt Project**.

Choose the **Qt GUI Main Window** application template (in the *Qt GUI* section). Note that if you select the templates for the dialog or widget, the following steps are the same.

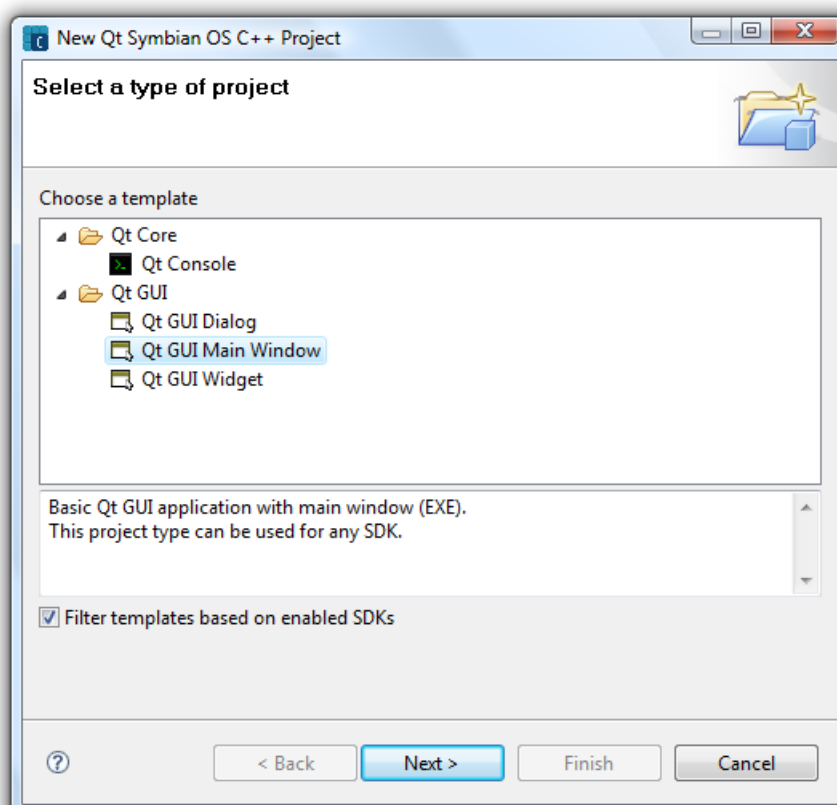


Figure 6: Carbide: New Qt Project Wizard

The **Next** page of the wizard is **New Qt Symbian OS C++ Project**. Define the project name – in this case *HelloWorldQt*. Keep the default location selected to create the project in your current workspace (note again, this must be on the same drive as the SDK and not contain spaces or other special characters).

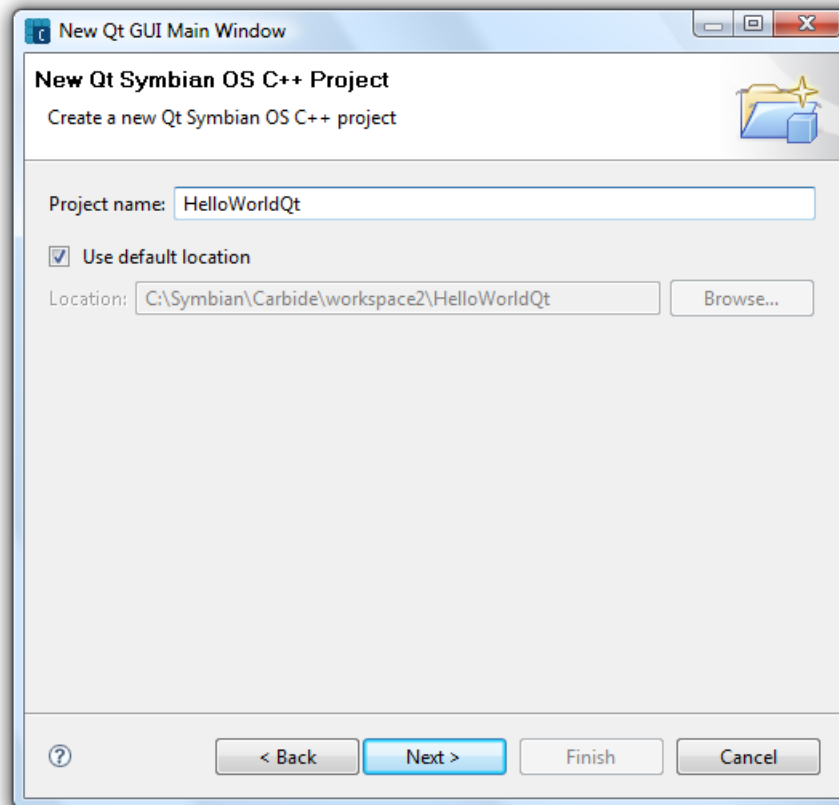


Figure 7: Carbide Qt Wizard: Project name and workspace

The **Next** page of the wizard is **Build Targets**. Choose the SDK(s) you want to use for building the project from among those installed to your PC. (You can add more SDKs to your project later on.) This should include a [Symbian platform SDK](#) that has been configured for use with Qt. At time of writing the only C++ Application Development SDK is the Symbian^1 SDK. (**Note:** this is a copy of the S60 5th Edition SDK v1.0.)

By default all build configurations/targets are selected. If you click next to the SDK you can select/deselect individual build targets:

- **Emulator Debug (WINSBW)** build binaries for the Windows-hosted Symbian platform emulator.
- **Phone Debug | Release (GCCE)** build binaries for the phone using the (free) GCCE compiler that was installed with the [SDK](#).
- **Phone Debug | Release (ARMV5)** builds binaries for the phone using the [ARM RealView Compiler \(RVCT\)](#). RVCT produces code that is a few percent smaller and faster than the current versions of GCCE supported for Symbian C++ development, but must be separately licensed from ARM. RVCT is primarily used by phone manufacturers to build binaries for device ROM.

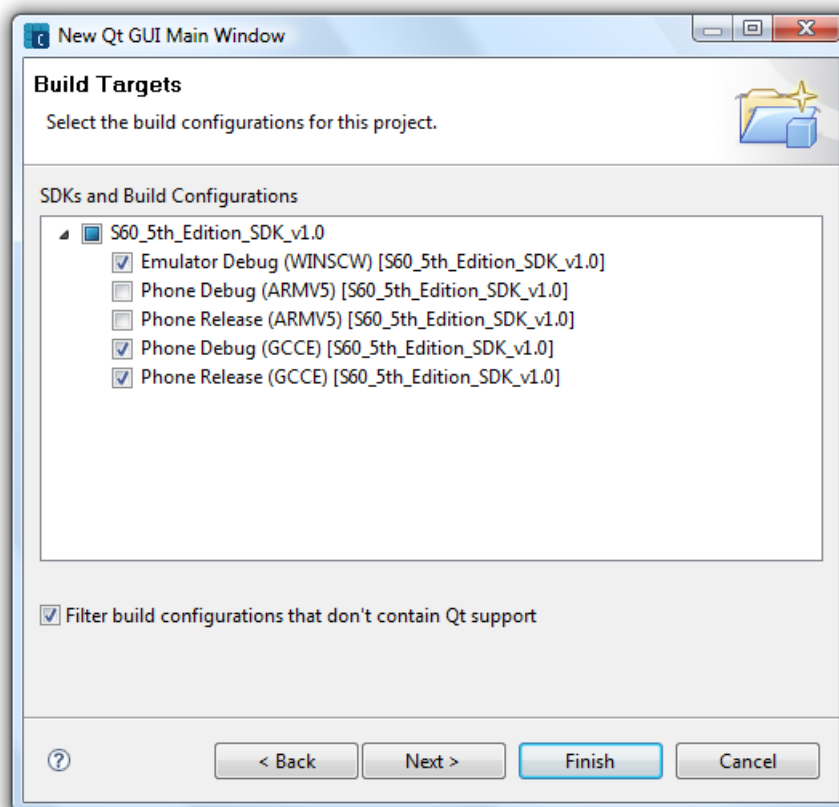


Figure 8: Carbide Qt Wizard: Build Targets

Most developers should de-select the ARMV5 options above as shown (the Emulator is needed by all developers, and GCCE is sufficient for most third-party development).

The **Next** page of the wizard sets the **Qt Modules**. The *Qt core* and *Qt GUI* modules are selected by default; these are all that are needed for this tutorial.

The **Next** page of the wizard is **BasicSettings**. This allows you to specify the name of the main window class, and to specify the application [unique identifier](#) (UID). Usually, you will leave the main window class unchanged.

The UID (actually the [SID](#), but for the moment we can ignore the distinction) defines the private area in the file system in which the application can store its data. Among other things the UID can also be used to programmatically identify and/or [start the application](#).

Carbide.c++ generates a random UID value for you starting with '0xE', which is the range of UIDs reserved for internal development and testing. If you want to release your application to the public, you need to get your own unique UID allocated by [Symbian Signed](#).

As we do not intend to release our Hello World Qt application to the public, we'll simply continue to use the value Carbide.c++ assigned us from the development range (you can change the UID later on, although you must be careful to change every instance of it found within your project).

Select **Finish** to close the wizard and create your application. Your workspace should look similar to the screenshot in Figure 11.

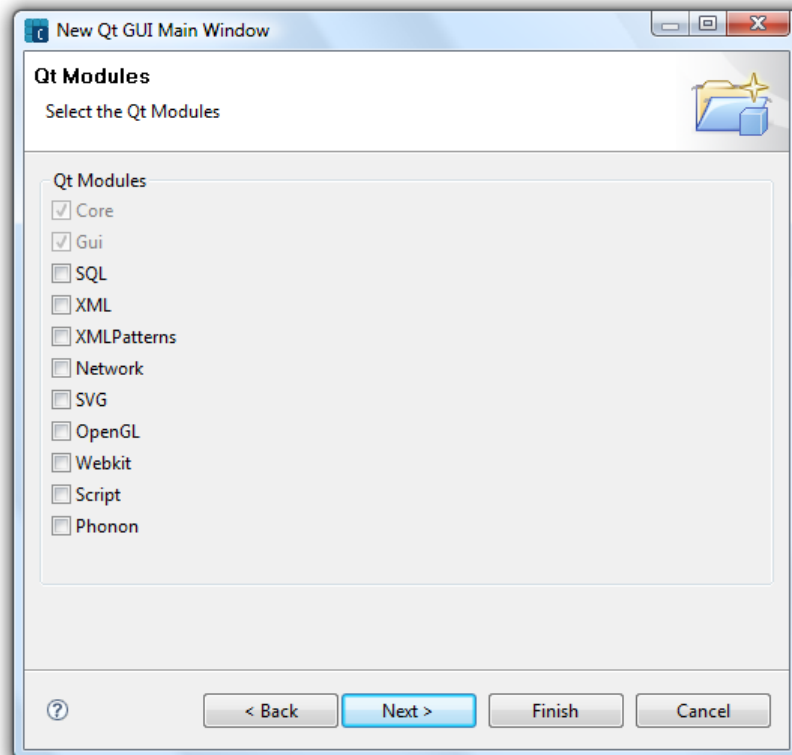


Figure 9: Carbide Qt Wizard: Qt Modules

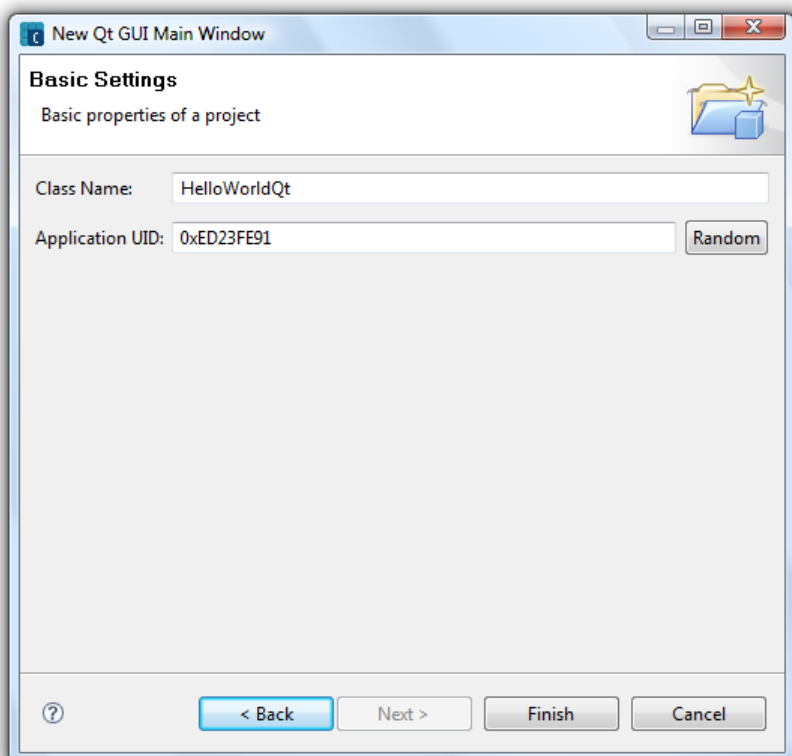


Figure 10: Carbide Qt Wizard: Basic Settings

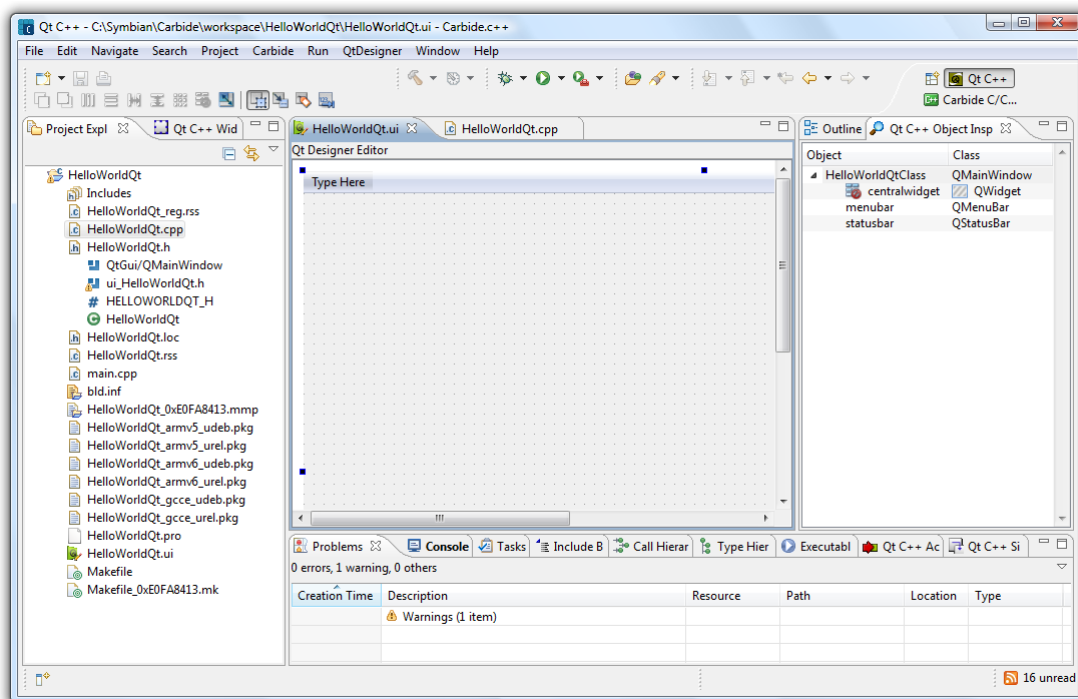


Figure 11: Carbide Workspace: Hello World Qt

The application should now build and run, displaying the project name in status area, and *Options* and *Exit* in the softkey area. You can skip ahead to the Targeting the Emulator section to build and run the application. Alternatively, true to the spirit of “HelloWorld” applications everywhere, below we show how to add a menu option and display a “HelloWorld” dialog when it is selected.

Launching A Hello World Message

Alternatively, true to the spirit of “Hello World” applications everywhere, below we show how to add a menu option and display a “Hello World” dialog when it is selected.

First we create a new action and add it to the main window menu bar. The menu action’s `triggered()` signal is connected to the `helloPressed()` slot.

```

HelloWorldQt::HelloWorldQt(QWidget *parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);

    // Add menu action to launch Hello World dialog

    // Create new action. The use of "tr" means this term is translatable.
    QAction *displayHello = new QAction(tr("Hello"), this);
    // Add the action to the window menubar (which automatically goes to the "Options" menu
    menubar()->addAction(displayHello);
    // Call the slot helloPressed() when the action is triggered.
    connect(displayHello, SIGNAL(triggered()), this, SLOT(helloPressed()));
}

```

The `helloPressed()` slot is declared in the class as shown below:

```
public slots:  
    void helloPressed();
```

The slot implementation to launch the dialog is trivial:


```
#include <QMessageBox> //for the message box  
...  
  
void HelloWorldQt::helloPressed()  
{  
    //Display information dialog with title "Hello" and text "World"  
    QMessageBox::information(this, tr("Hello"), tr("World!"));  
}
```

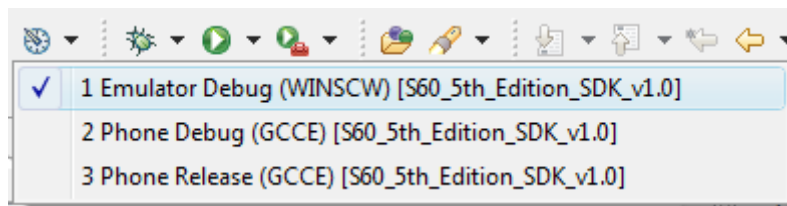
That's it!


2.4. Targeting the Emulator

Normally you'll start by building for the emulator; you can use the emulator for most of your development work (it is possible to access the Internet through the emulator, and even simulate GPS).

2.4.1. Building for the Emulator

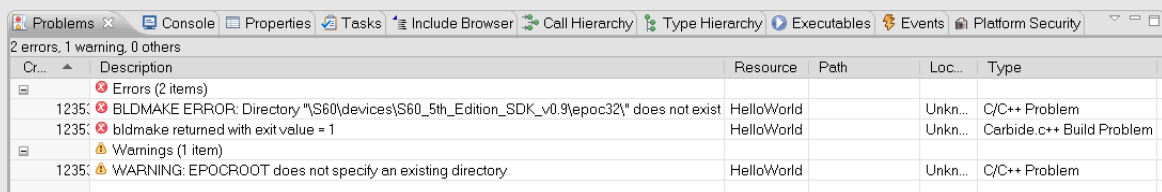
- First set the active build configuration. You can do this by clicking the **Build Configuration Tool** icon  in the toolbar or by selecting menu: **Project | Build Configurations | Set Active** and select **Emulator Debug**.



- Then build the current configuration using the **Build Tool** icon  in the toolbar or through the menu: **Project | Build Project** (You can also select a particular configuration to build from the Build icon selector).

There may be a number of build *warnings* (perhaps related to *Multiply defined Symbols*); these are expected and can be ignored.

Warning: If you get an error message similar to “**WARNING: EPOCROOT does not specify an existing directory**”, you did not place your workspace/project on the same drive as the application development SDK (which is installed by default to c:\). Delete the project and start again.



Cr...	Description	Resource	Path	Loc...	Type
2 errors, 1 warning, 0 others					
Errors (2 items)					
1235	BLDMAKE ERROR: Directory "S60\devices\S60_5th_Edition_SDK_v0.9\epoc32" does not exist	HelloWorld		Unkn...	C/C++ Problem
1235	blldmake returned with exit value = 1	HelloWorld		Unkn...	Carbide.c++ Build Problem
Warnings (1 item)					
1235	WARNING: EPOCROOT does not specify an existing directory	HelloWorld		Unkn...	C/C++ Problem

Carbide EPOCROOT Warnings

2.4.2. Running on the Emulator

If your application built successfully, click on the **Run** button  (Ctrl + F11). Upon the first launch of your project, Carbide.c++ will ask you which executable you want to launch:

- If you choose *HelloWorldQt.exe*, the emulator will be launched and your application started automatically. The emulator will close once you exit your application.
- If you choose *Emulator* the emulator (epoc.exe) will be launched and you will need to navigate to the application and start it by clicking on the icon (just as you do when starting an application on a device). We'll explain how to find the application shortly.

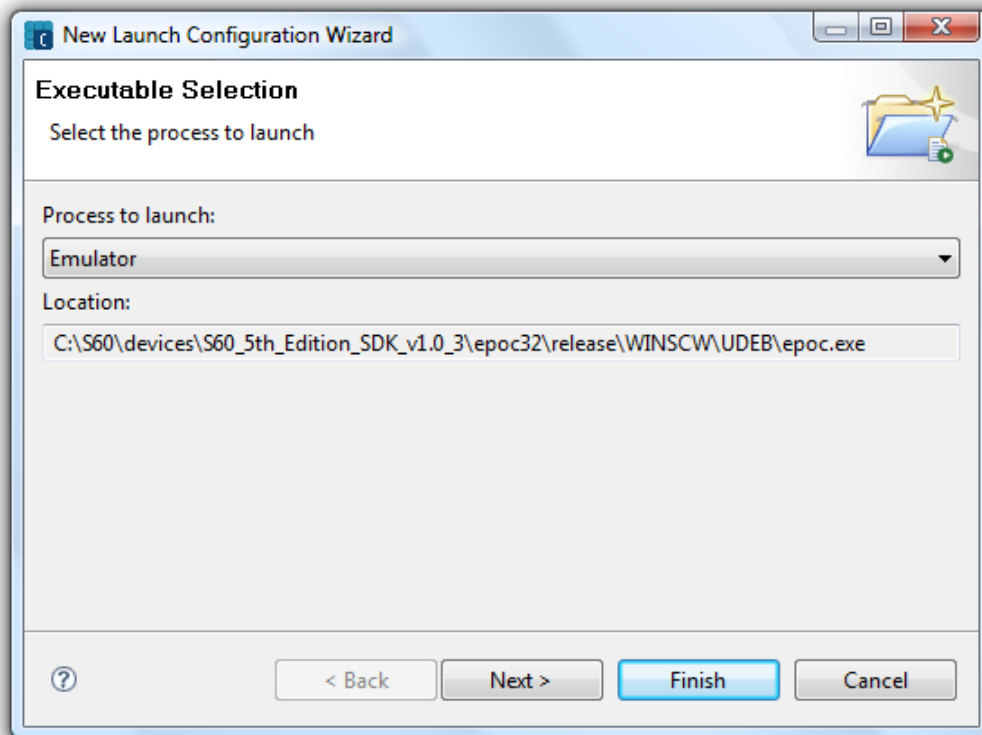



Figure 12: Target Executable Selection

It may sound more difficult to choose the second method, but it has some advantages. You can leave the emulator running if you are only doing small edits in your source code – simply close your application in the emulator, recompile and restart your app through the emulator’s menu. You’ll also see any error messages that may be shown when you exit the application, because the emulator will not shut down instantly after you exit Hello World in the emulator. Those error messages are also visible in Carbide.c++’s console window.

When the emulator starts for the first time you might have to wait for several minutes before it is completely ready for use. Successive starts will be a lot faster, because Windows caches most of the emulator DLLs. You may also need to register your SDK with forum Nokia - this takes a couple of minutes and can be done online (you will have to register with Forum Nokia).

If you decide to launch the emulator and navigate to your application: First, open the menu through the **menu** symbol on the bottom left of the screen. Your own application will be located at the bottom of the **Applications** folder; use your mouse to navigate in the emulator’s menus. The application will appear as shown in the right-hand image when it is launched.

2.4.3. Debugging on the Emulator

The Emulator is the default debug target - you simply click the **Debug** button .

Debugging on the Emulator is not covered further in this tutorial. See **Carbide.c++ User Guide > Debugging projects** for extensive information on debugging using Carbide.c++.

Figure 13 shows how your application should look when it is launched.

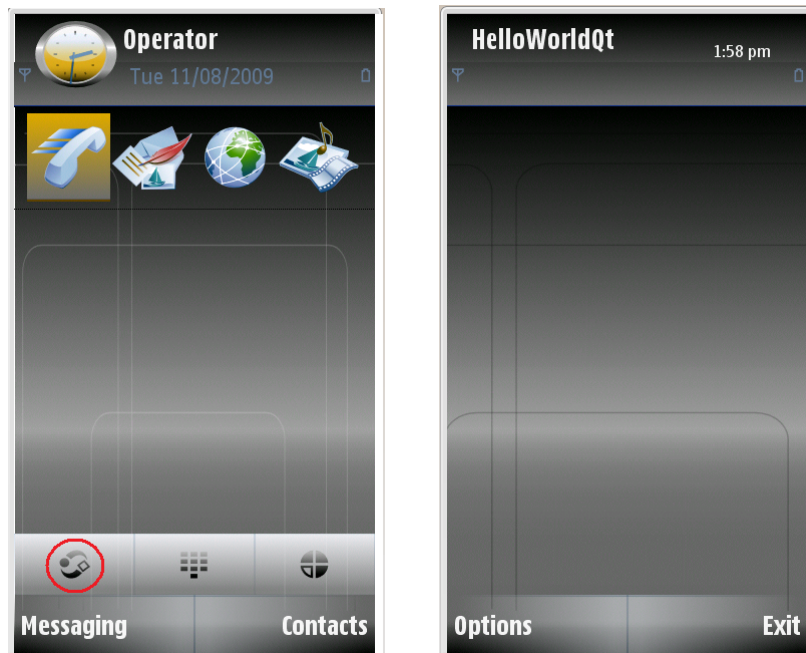


Figure 13: Menu Symbol (left) and HelloWorldQt Default View (right)


2.5. Targeting the Device

The emulator can be used for most of your development work. However, some situations still require a real device – for example, when you want to use the camera or the acceleration sensor.

Tip: You should test your applications on the phone from time to time, even if it is fully supported by the emulator.

When you’ve finished development, you’ll also want to build a release version; stripping out debug code and symbol information to make your binaries smaller and more efficient.

2.5.1. Building for the Device

To tell the IDE that you want to build for the device, change the active build configuration to a phone-release configuration for GCCE (unless you have the [RVCT compiler](#)). As before, use the **Build Configuration Tool** toolbar icon () to select the active-build configuration.

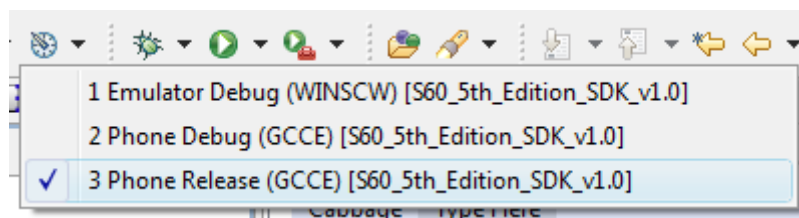



Figure 14: Carbide: Select Release Target

Next, choose to build the current configuration using the toolbar **Build** icon  (or in the menu: **Project | Build Project**).

This will automatically compile the release project using the GCC compiler and create a self-signed installation file called `HelloWorldQt2_gcce_urel.sisx` in your project's workspace root directory. You now need to transfer this file to your phone to install it. Note that there is an unsigned version of the file `HelloWorldQt2_gcce_urel.sis` created as well – this will usually not be installable, so take care not to select it instead! Tip|Use Carbine.c++ to find the file on your PC. Navigate to the file in the project view, then right-click on it and select **Show in Explorer**. Don't forget to switch back to the **Emulator Debug** build configuration when you continue development!

2.5.2. Installing on the Device

You can use the PC Suite that came with your phone to install the application on your device

- Ensure that the PC Suite is installed and running.
- Connect your device to the PC via Bluetooth or USB and add the phone to the known devices in the PC Suite (if necessary).
- Double-click the `.sisx` file in Windows Explorer or the Project Explorer window of Carbine.c++.

If the PC Suite is not installed on your PC, you can send the file to the phone via Bluetooth or IrDA (if available):

- Locate the `.sisx` file in Windows Explorer.
- Right-click on it and select **Send to | Bluetooth device**.

You will be prompted to install the application when you open the message.

Warning: If you get a *Certificate Error* message when you try to install the application, then it is possible your phone has been configured to prevent installation of self-signed sis files. To change this behavior, go to **Settings - Application manager - Installation settings - Software installation** and change the setting from **Signed Only** to **All**. Another alternative is that your PC clock is set later than the phone clock, which makes the certificate invalid. For other errors received upon installation, consult the [installation error troubleshooting guide](#).

2.5.3. Debugging on the Device

Debugging on a production phone is covered in the topic: [Getting Started with Debugging on the Device](#).

3. Summary

In this tutorial you set up your Qt development environment, learned how to create a skeleton application using Carbide.c++ *Qt Project* wizard, and how to get it up and running on both the Symbian platform emulator and on the device.

4. Related Information

Further reading:

- [Qt Technical Overview](#)
- [Qt Q&As](#)
- [Symbian pre-releases site](#)
- [Qt Reference Documentation](#) (recommended)
- [Qt Developer's Library](#) (Forum Nokia)
- [Qt on Samsung Symbian Part 1](#) and [Part 2](#)
- [A Video Guide for Setting up Qt development environment for Symbian](#)

About the Kits:

- [What are the Kits?](#) explains the [ADT](#) and [SDK](#).
- [Kits Q&As](#)
- [Getting Started with Debugging on the Device](#)
- [Symbian^1 SDK Release Notes](#)
- [Symbian^1 SDK Installation Guide](#)

Nokia, the Nokia logo, Qt, and the Qt logo are trademarks of Nokia Corporation and/or its subsidiary(-ies) in Finland and other countries. Additional company and product names are the property of their respective owners and may be trademarks or registered trademarks of the individual companies and are respectfully acknowledged. For its Qt products, Nokia operates a policy of continuous development. Therefore, we reserve the right to make changes and improvements to any of the products described herein without prior notice. All information contained herein is based upon the best information available at the time of publication. No warranty, express or implied, is made about the accuracy and/or quality of the information provided herein. Under no circumstances shall Nokia Corporation be responsible for any loss of data or income or any direct, special, incidental, consequential or indirect damages whatsoever.

Copyright © 2009 Nokia Corporation and/or its subsidiary(-ies).



This document is licensed under the [Creative Commons Attribution-Share Alike 2.5](http://creativecommons.org/licenses/by-sa/2.5/legalcode) license.

For more information, see <http://creativecommons.org/licenses/by-sa/2.5/legalcode> for the full terms of the license.

About Qt

Qt is a cross-platform application framework. Using Qt, you can develop applications and user interfaces once, and deploy them across many desktop and embedded operating systems without rewriting the source code. Qt Development Frameworks, formerly Trolltech, was acquired by Nokia in June 2008. For more details about Qt please visit <http://qt.nokia.com>.

About Nokia

Nokia is the world leader in mobility, driving the transformation and growth of the converging Internet and communications industries. We make a wide range of mobile devices with services and software that enable people to experience music, navigation, video, television, imaging, games, business mobility and more. Developing and growing our offering of consumer Internet services, as well as our enterprise solutions and software, is a key area of focus. We also provide equipment, solutions and services for communications networks through Nokia Siemens Networks.

**NOKIA**