# Archived:Qt Carbide.c++ IDE Quick Start

**Archived:** This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Qt SDKs that can work directly with standalone Symbian SDKs are no longer supplied, and Carbide.c++ does not at time of writing integrate with the Qt SDK. Developers should instead see: Using Symbian C++ in the Qt SDK

This Quick Start is relevant if you want to create applications with Qt using the Carbide.c++ IDE. It assumes that you have already followed the instructions in Archived:Using Qt with Standalone SDKs to set up your (command line) development environment.

The article 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.

**Comes with Code:** File:HelloWorldQt.zip

**Note:** This article applies to:
- Carbide.c++ 2.3.0
- Qt 4.7.0 (note that some of the following screen shots show Qt 4.6.2 SDK, however the behaviour is identical)
- Nokia Smart Installer 1.0.0
- Symbian^1 SDK (any standalone Symbian SDK)

It does not cover using Carbide.c++ with the Qt SDK

## Installing Carbide.c++

Carbide.c++ was installed from the Application Developer Toolkit when you followed the instructions in Archived:Using Qt with Standalone SDKs.

In summary

- Download and install the Carbide.c++
- Extract the Carbide.c++ windows compiler patch file into the `\x86Build` directory under your Carbide installation, e.g. `C:\Symbian\Tools\ADT_1.4\Carbide.c++\x86Build\`
- (Optional) Configure the toolchain to allow command line building for the emulator using the Windows **start** button: **All Programs | Symbian Foundation ADT v1.4 | Carbide.c++ | Configure environment for WINSCW command line**

## Starting Carbide.c++

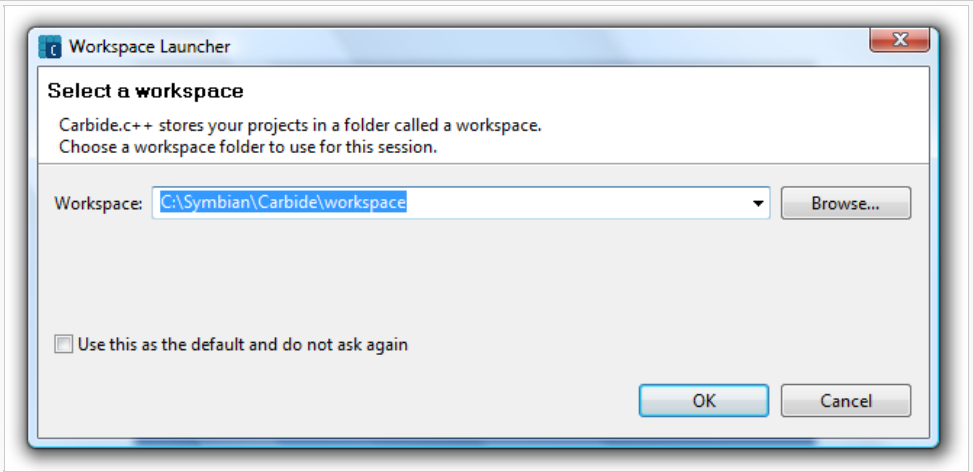
Carbide.c++ is launched from the Windows **start** button:

- **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 drive `C:\`, an example of a correct workspace path is: `C:\Symbian\development\`.

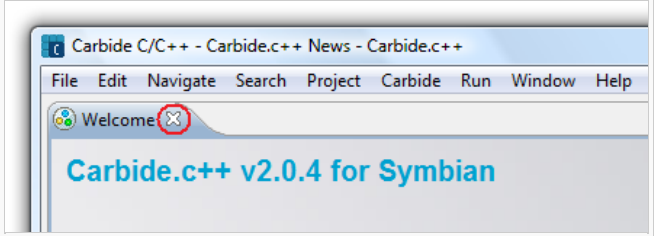
**Note:** Your Symbian workspace must be on the drive where you installed your SDK. You must also ensure that the path name of the workspace does not contain non-alphanumeric characters or spaces. This is because the Symbian toolchain uses command line tools that cannot read special path names.



Carbide: Workspace Launcher

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


(C) Copyright Nokia 2013. All rights reserved.

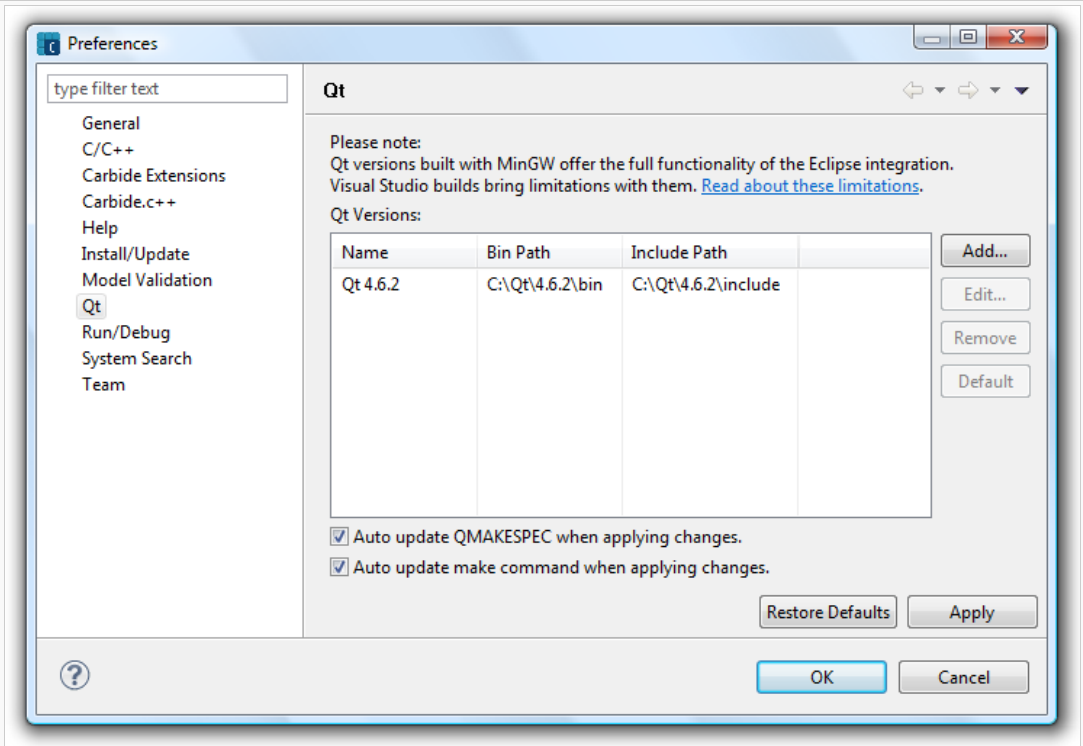


Carbide: Welcome Tab

## 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**).



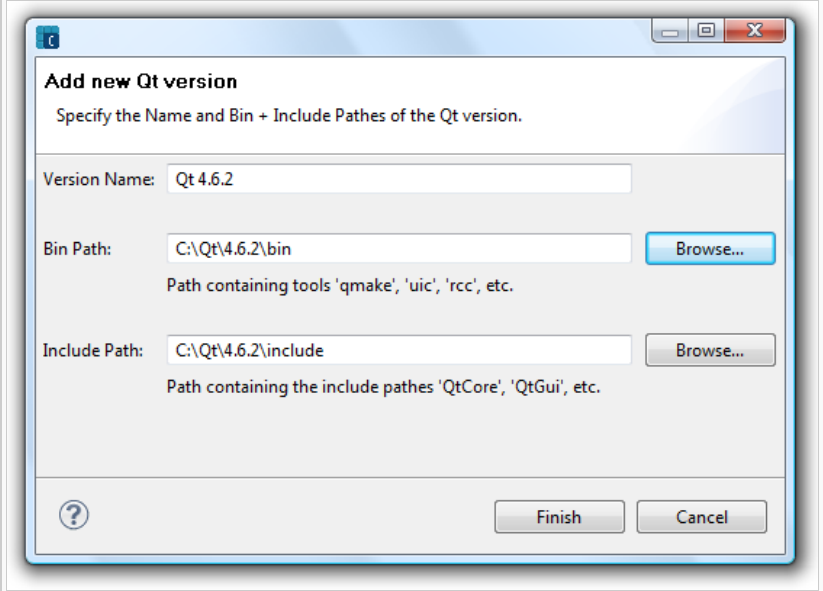
Carbide Preferences for Qt

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.

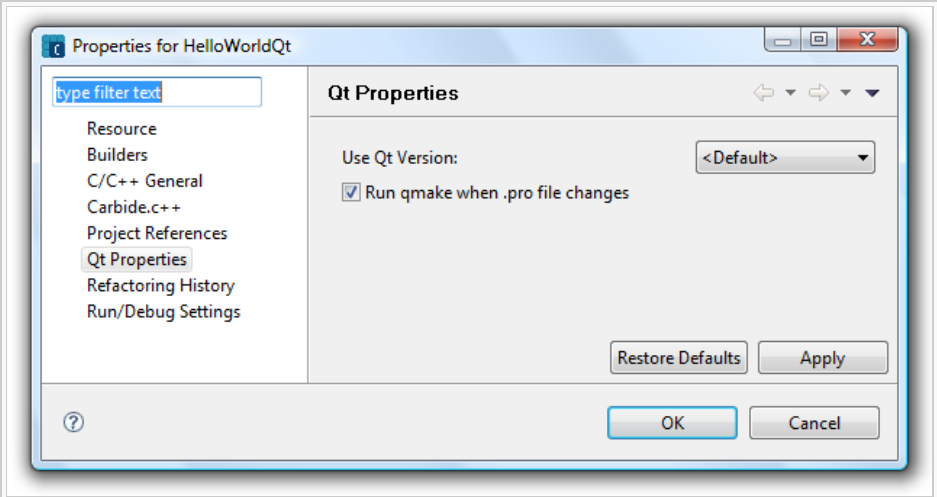
- Enter the name of the Qt release
- Enter the full path to the Qt `\bin` directory
- Enter the full path of the Qt `\include` directory (this will be seeded with the correct value based on your specified `\bin`)
- Press **Finish** to save the dialog.


(C) Copyright Nokia 2013. All rights reserved.




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**




Carbide Project properties for Qt

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.

# Qt Carbide.c++ Helloworld

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

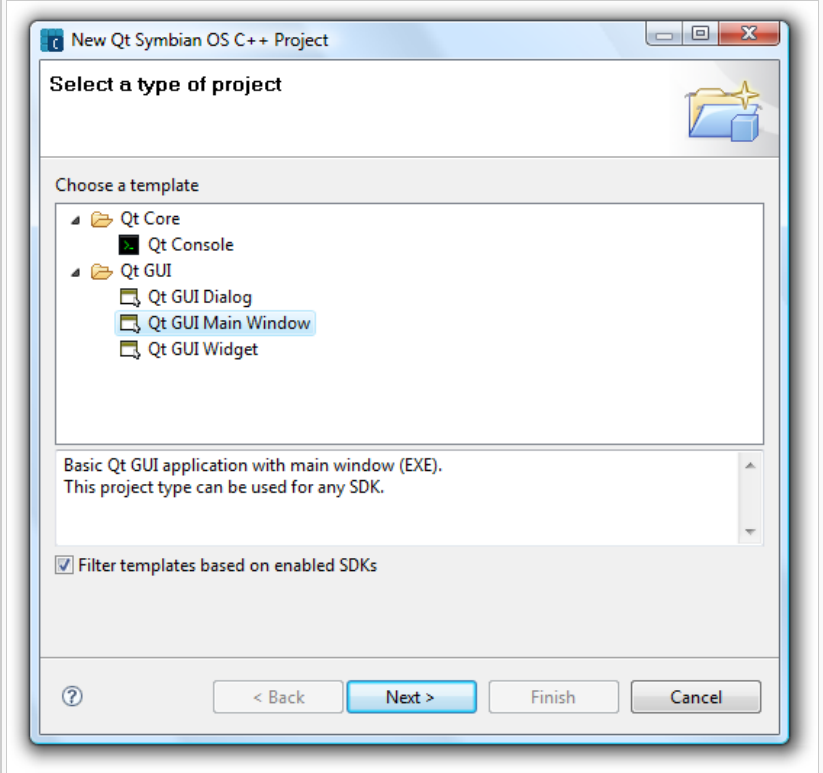
This tutorial is not intended as a lesson in Qt development (although you may learn a little along the way)! It is designed to familiarise you with the Carbide.c++ IDE, and how you get your application onto the phone. If the project builds cleanly and installs then this is the best verification possible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

## Creating a Project

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

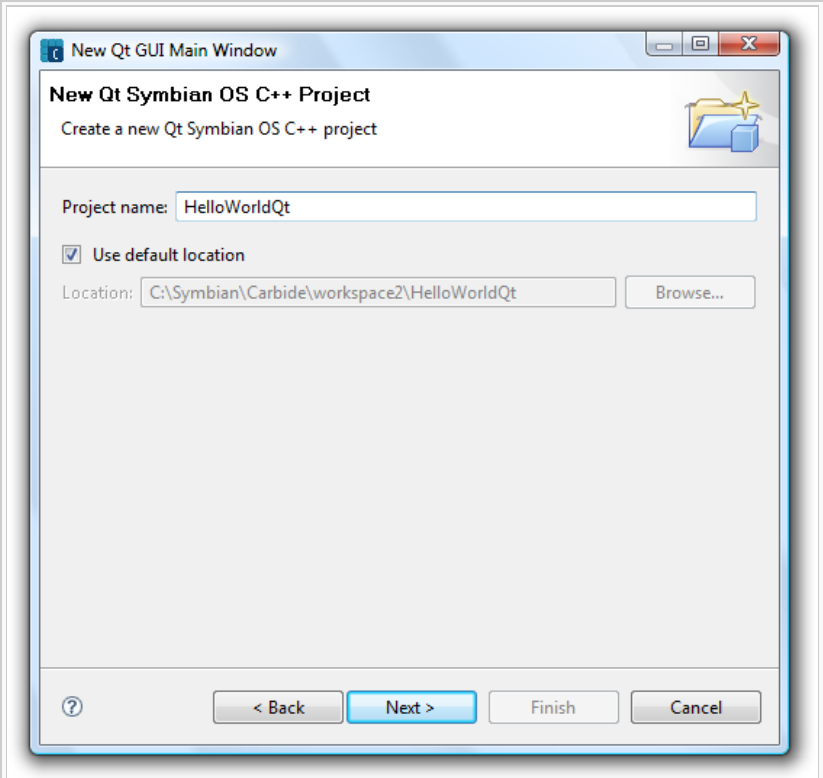
Choose the **Qt GUI Main Window** application template (in the *Qt GUI* section). Note that if you select the templates for the dialog or widget, the following steps are the same.


(C) Copyright Nokia 2013. All rights reserved.




Carbide: New Qt Project Wizard

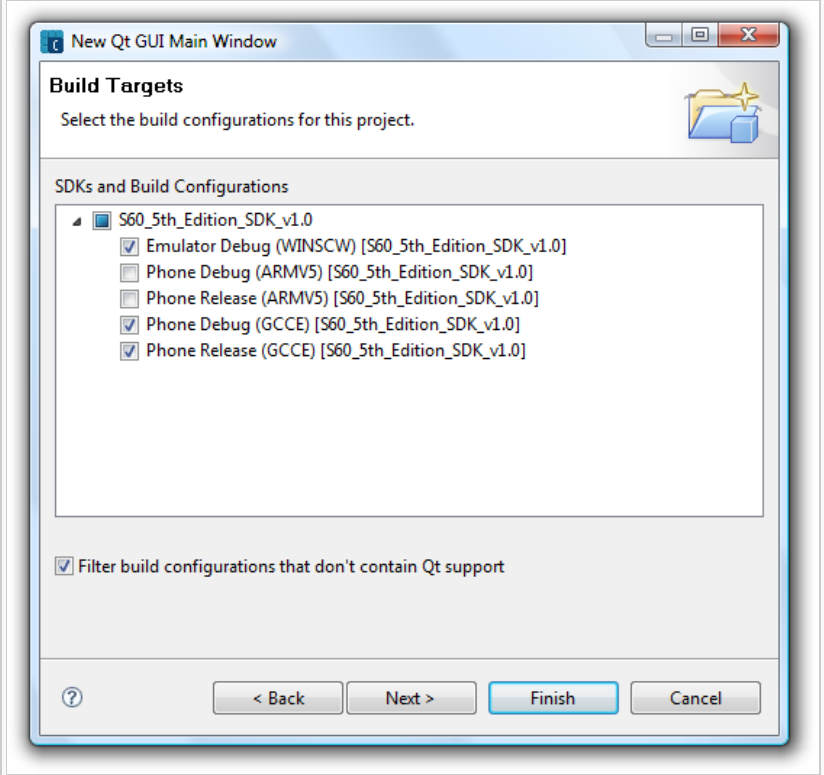
The **Next** page of the wizard is **New Qt Symbian OS C++ Project**. Define the project name - in this case *HelloWorldQt*. Keep the default location selected to create the project in your current workspace (note again, this must be on the same drive as the SDK and not contain spaces or other special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




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).


(C) Copyright Nokia 2013. All rights reserved.



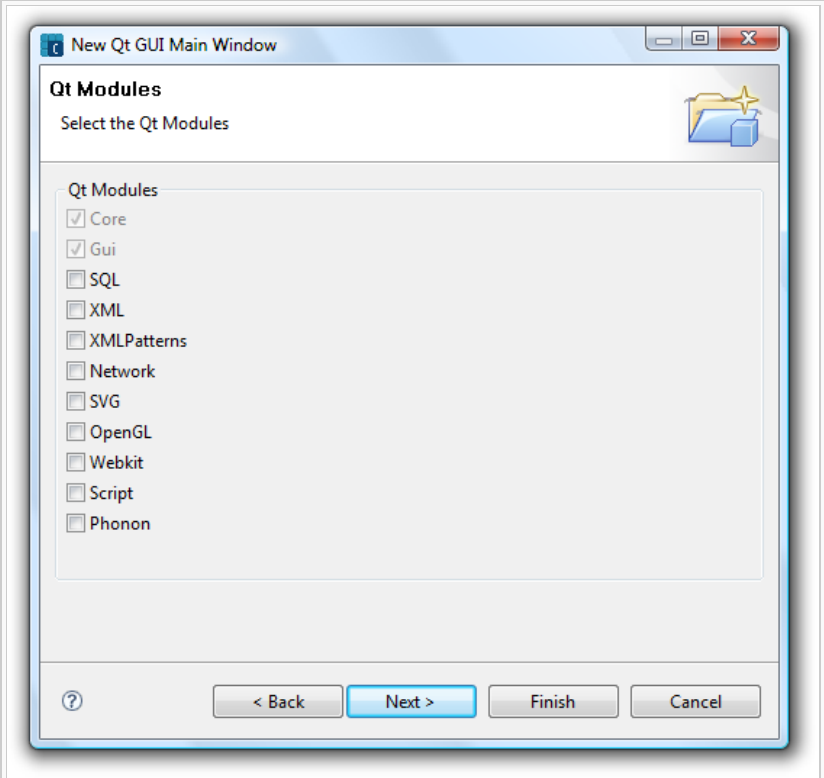

Carbide Qt Wizard: Build Targets

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

- **Emulator Debug (WINSCW)** 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.

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).

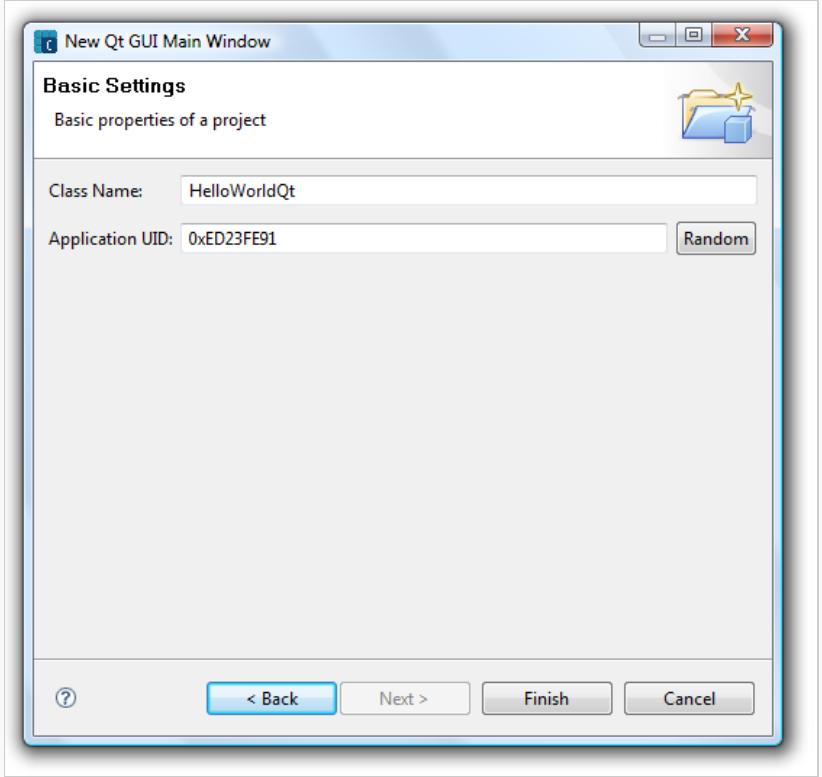
The **Next** page of the wizard sets the **Qt Modules**. The *Qt core* and *Qt GUI* modules are selected by default; these are all that are needed for this tutorial.




Carbide Qt Wizard: Qt Modules

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.


(C) Copyright Nokia 2013. All rights reserved.



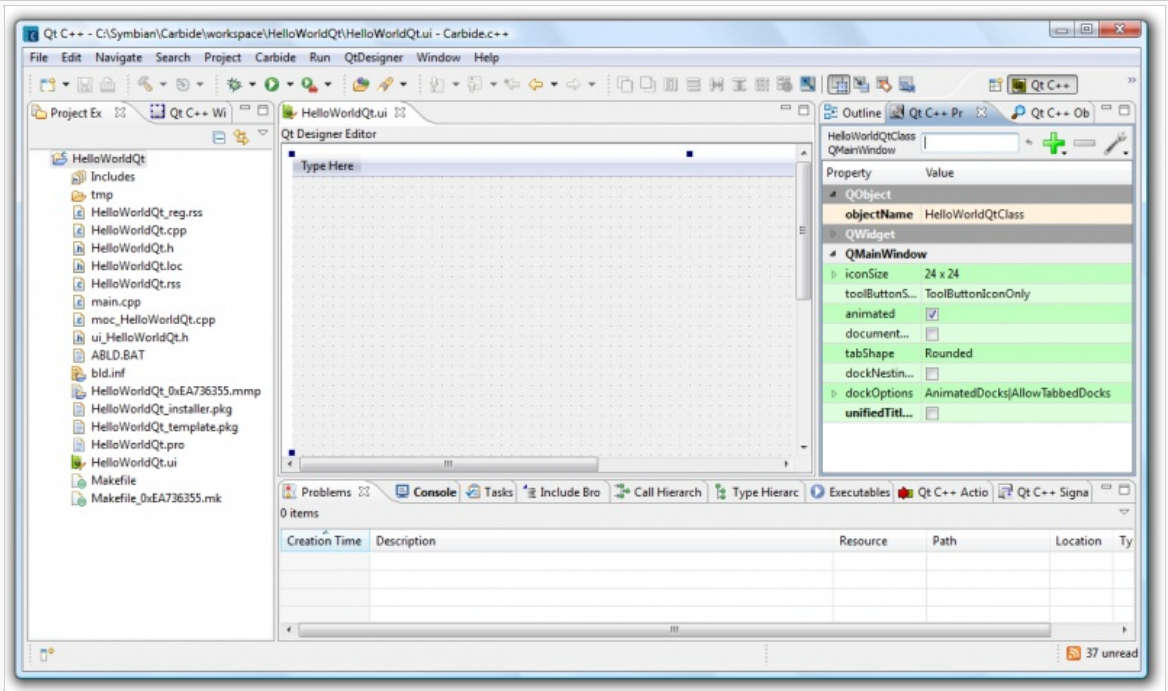

Carbide Qt Wizard: Basic Settings

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 below.




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 next 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**

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)
```


(C) Copyright Nokia 2013. All rights reserved.

```
    : QMainWindow(parent)
{
    ui.setupUi(this);

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

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()
{
    QMessageBox::information(this, tr("Hello"), tr("World!")); //Display information dialog with title "Hello" and text "World"
}
```

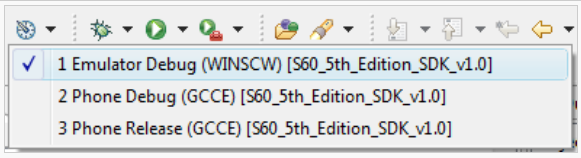
That's it!

## 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).

### 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**.



Carbide: Set Active Build Configuration

- 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.

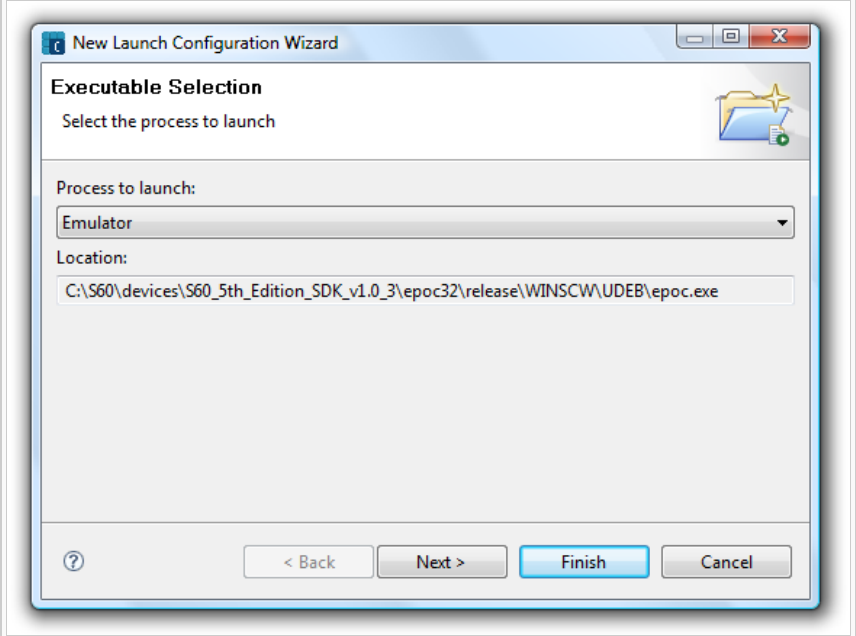
**Warning:** If you get an error message similar to "**WARNING: EPOCROOT does not specify an existing directory**", you did not place your workspace/project on the same drive as the application development SDK (which is installed by default to C:\). Delete the project and start again.
Expand the thumbnail for how this might appear in Carbide.c++:

Carbide EPOCROOT Warnings

### 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:


(C) Copyright Nokia 2013. All rights reserved.




Target Executable Selection

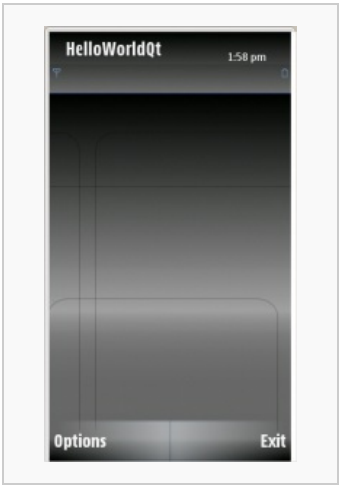
- If you choose *HelloWorldQt.exe*, the emulator will be launched and your application started automatically. The emulator will close once you exit your application.
- If you choose *Emulator* the emulator (epoc.exe) will be launched and you will need to navigate to the application and start it by clicking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

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 Nokia Developer - this takes a couple of minutes and can be done online.

*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.

Menu Symbol




HelloWorldQt Default View

## 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++.

## 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.


(C) Copyright Nokia 2013. All rights reserved.

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.

## Specifying the installation package

Symbian platform SIS installation files (`.sis`) are created from package file (`.pkg`) definitions. SIS file are often then digitally signed (this is a requirement on most phones) and renamed with the file extension `.sisx`.

Qt package files are created automatically from the content of the Qt project file (`.pro`). By default two files are created *HelloWorldQt_template.pkg* and *HelloWorldQt_installer.pkg*. The *_template.pkg* file contains all files needed by the application except for Qt itself. The installation file produced is not particularly suitable for product release because you can't assume that Qt was already present on the device. Its possible to modify the Qt project file to include the Qt binaries in this package - which makes the SIS file much bigger, but guarantees that it is present on the device.

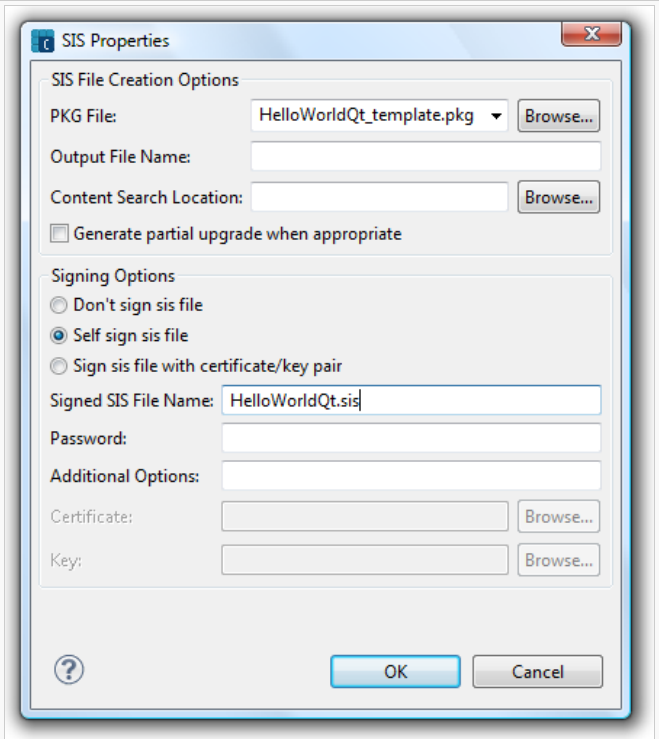
An alternative approach is the use of a "smart installer". In this case the delivered SIS file contains both the application SIS and a small smart installer SIS; if Qt is not present on the device it is automatically installed before the application. The *HelloWorldQt_installer.pkg* defines this wrapper, and must be built after the application SIS.

> **Warning:** The smart installer is not in Qt SDKs by default (except the Nokia Qt SDK). See Nokia Smart Installer for Symbian for more information on the smart installer, and where it can be downloaded

Symbian platform deployment options are discussed more thoroughly in the article Deploying a Qt Application on Symbian.

To tell Carbide which package files you want it to use as part of the build process:

- Open the Project properties dialog (select the project folder in **Project Explorer** and then do menu **File** | **Properties** (or **right click** | **Properties**))
- Navigate down to **Carbide.c++** | **Build Configurations** | **SIS Builder** tab
- Check that the "Active Configuration" is the one you want (GCCE or ARMv5, Debug or Release)
- Press the **Add** button to specify a package file (.pkg) to be build for the current active configuration.



Carbide: SIS Properties

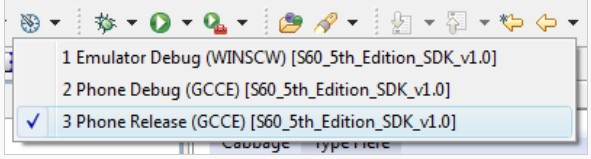
Select the PKG File you want to build from those in the project directory (first *HelloWorldQt_template.pkg*). The default values for the other settings will create a sis file "HelloWorldQt_template.sis" and a self-signed SIS file "HelloWorldQt_template.sisx" in the current directory.

You can change the name of the output SIS and SISX files in the fields provided. Here we have changed the output signed file to *HelloWorldQt.sis* because that is the filename that the smart installer package file expects.

Press **OK** to save the SIS builder definition and then add another one for *HelloWorldQt_installer.pkg* (there is no need to do anything other than select the package file).

## 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.



Carbide: Select Release Target


(C) Copyright Nokia 2013. All rights reserved.

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 GCCE compiler and create any SIS files you specified in your SIS builder specification.

You now need to transfer the `HelloWorldQt_installer.sisx` file to your phone to install it (see next section). Note that there is an unsigned version of the file `HelloWorldQt_installer.sis` created as well - this will usually not be installable, so take care not to select it by accident!

> **Tip:** Use Carbide.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!

### Installing on the device

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

- Ensure that the 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 suite (if necessary).
- Double-click the `.sisx` file in Windows Explorer or the Project Explorer window of Carbide.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 its 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**.
>
> It is also possible that your PC clock is set later than the phone clock, which makes the certificate invalid.

### Debugging on the device

Debugging on a production phone is covered in the topic: Carbide.c++ On-device Debugging Quick Start.

## Troubleshooting

The majority of users will have followed the above instructions to create and run the HelloWorld application. If you do encounter development environment issues, then you should:

- Review the Qt Known Issues (on Gitorious).

## Summary

In this tutorial you learned how to configure Carbide.c++ for Qt development, 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.

## Related Information

Further reading:

- Category:Carbide.c++
- Carbide.c++ On-device Debugging Quick Start

- Qt Reference Documentation (recommended)
- Qt Developer's Library (Nokia Developer)
- Qt on Samsung Symbian Part 1 and Part 2
- A Video Guide for Setting up Qt development environment for Symbian
- Qt Creator Quick Start

© 2009 Nokia Corporation and/or its subsidiary(-ies). This document is licensed under the Creative Commons Attribution-Share Alike 2.5 license. See http://creativecommons.org/licenses/by-sa/2.5/legalcode for the full terms of the license.


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.