

# Using Carbide.c++



**This article needs to be updated:** If you found this article useful, please fix the problems below then delete the `{{ArticleNeedsUpdate}}` template from the article to remove this warning.

**Reasons:** hamishwillee (25 Aug 2011)

The original text is based on a Symbian Press booklet and was updated for the v2.0.4 release. Carbide.c++ is now at v3.2 (or later) so this article is due a review and possible refresh.



**Note:** The original text is based on a Symbian Press booklet and was updated for the v2.0.4 release.

This booklet provides an introduction to the Carbide.c++ Integrated Development Environment (IDE) for Symbian C++ applications, available from [here](#).

## The Eclipse Platform

Carbide.c++ is based on the Eclipse platform, which was launched in November 2001 by IBM and seven other companies as an open source project with the aim of creating a free multi-platform IDE. Maintained by the non-profit Eclipse Foundation organization ([www.eclipse.org](http://www.eclipse.org)) and its member companies (including Symbian and Nokia), Eclipse has grown into an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software. As a result, a growing number of companies are producing commercial or free products which are based on the Eclipse technology, one example being Carbide C++.

The real goal for Eclipse is an environment that makes the integration of various types of tools easy. Eclipse fulfils this requirement by using an architecture that implements a large set of reusable frameworks that can be used by any tool. It contains a framework that implements a consistent user interface, a framework for organizing and accessing programming artefacts from the file system, frameworks for text editors, application builders, debugging sessions, team programming, version control, and so on. The most important aspect of Eclipse's architecture is the fact that new code can easily be integrated into the platform as pluggable units of functionality. Of course, support for pluggable code is not unique to eclipse since all major IDEs today support extensions, but the unique aspect of Eclipse is that there is no distinction between the core frameworks and those added by various developers in order to extend the basic functionality.



## The Carbide.c++ C/C++ IDE

The **Carbide.c++ IDE** is based on the Eclipse IDE version 3.4 (Ganymede) with additional plug-ins that enable Eclipse to understand how to handle Symbian projects. Since Carbide.c++ is based on the Eclipse IDE, it can be further customized with the installation of additional commercial or free plug-ins, such as a version control system, a UML modelling system, and so on.

The installation of Carbide.c++ is simple because it comes with its own Windows-based installer. After installing and launching the product, the developer is presented with a typical IDE which is largely identical to the standard Eclipse IDE and is shown in Figure 1. Developers familiar with Eclipse itself or other Eclipse-based products, will find Carbide.c++ easy to use because the Carbide.c++ plug-ins follow the design philosophy of Eclipse and its frameworks.

Since Carbide.c++ is designed to handle Symbian C++ applications, by default, the left hand side of the IDE displays the C/C++ *Project Explorer* view, which is designed to show the C++ source, header and resource files that comprise a technical Symbian C++ project.

As figure 2 shows, below the view there is the *Symbian Project Navigator* view, which displays the structure of the Symbian C++ project in the manner that is described in a typical Symbian *MMP* file, referenced by a *bid.inf* file.

Developers familiar with the manual creation of such files, as well as with CodeWarrior for Symbian Platform, will find this view easier to understand.

Next is the *Remote Connections* view which provides the ability to monitor, create, edit, or remove common connection settings for remote agents. Very handy for

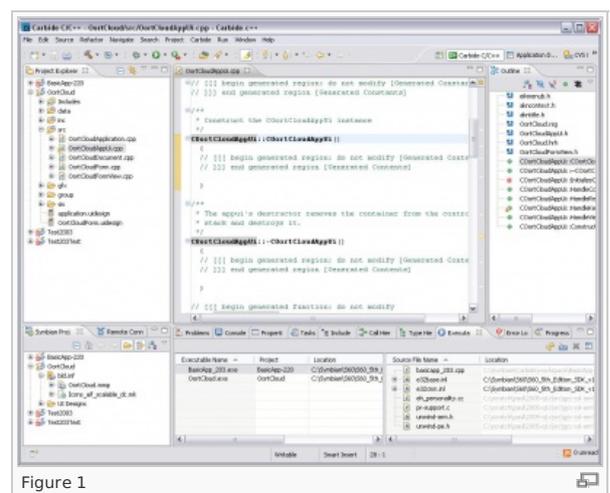


Figure 1

edit, or remove common connection settings for remote agents. Very handy for targeting applications being debugged on different target devices using varied connections settings.

In the middle of the IDE there is the C/C++ source editor. Multiple editor windows can be opened simultaneously and the developer can switch from one to another by clicking on the top tab of the required editor window, or setting up their own preferred keyboard shortcut.

Next, on the top right hand side of the IDE there are two view opened by default: the *Outline view* and the *Help view*.

- The *Outline view* shows the referenced header files and the functions defined in the C/C++ source file currently selected in the editor.
- The *Help view* displays help information concerning the element currently selected with the mouse anywhere on the IDE. It also allows navigation to read any help topic related to the Carbide.c++ and its features.

Finally, along the bottom of the IDE are a host of status and other useful views, including:

- *Call Hierarchy view* - explore call-graphs by means of a tree
- *Console view* - shows the output of a process and allows you to provide keyboard input to a process
- *Executables view* - provides a dynamic list of executables and their related source files
- *Include Browser view* - visualizes the include relations among files
- *Problems view* - displays build errors
- *Properties view* - displays property names and basic properties of a selected resource
- *Tasks view* - displays tasks that you manually add
- *Type Hierarchy view* - presents inheritance relations and members of type

The collection of these windows that automatically appear when a Symbian C++ project is loaded, combined with their location in the IDE, from the Carbide.c++ C/C++ perspective, denoted with a clickable icon on the top-right section of the IDE.

Carbide.c++ comes with various perspectives predefined for different development phases. So, for example, there is a Debug perspective that when selected opens a different set of windows and views on the IDE which are suitable for debugging sessions.

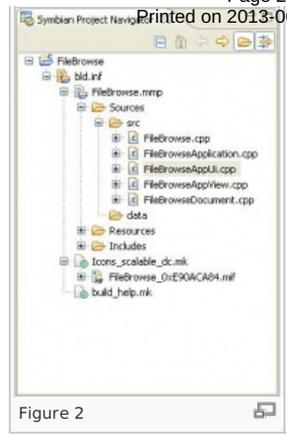


Figure 2



**Note:** Each developer can customize the windows and views associated with each perspective to their liking and that Carbide.c++ can recall these settings when re-launched.

The current perspective can be changed by using the **Window > Open Perspective** option. The layout of the currently open perspective will be saved and the new perspective will be opened. Views which are not currently visible can be opened with the **Windows > Show View** option. The view will be opened at the location that it was previously open in the current perspective, or in its default location if it has never been opened in the current perspective.



Figure 3

## Importing existing Symbian C++ projects

Some Symbian C++ developers may already have various Symbian projects on their computer and have been using either CodeWarrior for Symbian Platform or the command-line tool-chain to build them. They will find that Carbide.c++ provides a facility to easily import their projects without having to create a new project from scratch and then manually insert the source files (although it is also possible to do this). The Eclipse platform supports the creation of wizards to allow the import and export of information and files from the IDE. This facility is used by Carbide.c++, and as a result, a Symbian bld.inf import wizard has been implemented. As the Figure shows, by selecting **File > Import** menu item, the developer launches an import panel which has various options, amongst which is the option to import a bld.inf file as shown in figure 5. After selecting this option, the developer is asked to select the bld.inf file to import from the host computer's hard dis(s).

When this step is completed, Carbide.c++ displays all the Symbian Platform SDKs that are installed on the host PC (known as autodetection) and asks the developer to select the one appropriate for the project, as shown in figure 6. In the next step, the developer can select which of the Symbian MMP files referenced in the imported bld.inf file should be used in order to parse the appropriate MMP file(s). Finally, the developer can select the project name and the root directory for the project. At the end of this wizard's operation, a Symbian C++ project is opened and ready for development in Carbide.c++.

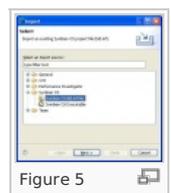


Figure 5

## Creating a new Symbian C++ project

Carbide.c++ contains a wizard which allows the creation of a new Symbian C++ project from scratch. The wizard can be launched by using the **File > New Project** menu item, as shown in figure 9.



Figure 6

Carbide.c++ contains specialized templates for various project types shown in figure 10. using one of these templates will automatically generate a new project and at the same time generate source files with some code, depending on the type of project selected by the developer.

Since Carbide.c++ automatically detects the installed Symbian Platform SDKs on the host PC, the project templates can be filtered based on the detected SDKs.

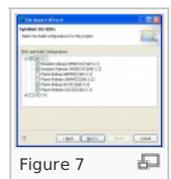


Figure 7

The next step of this wizard asks for the directory in which to save the project files and then for the build configurations to be enabled, as shown in Figure 12. The final steps of the wizard allow the developer to specify information such as the UID of the application (although this can be changed at a later stage) and the directories to be used for the various types of files which will be created. This completes the operations of the wizard and results in a new project with some initial files which can be further developed.



Figure 8

## Project Properties

Several project-related parameters can be modified by launching the project properties window. This window can be launched by

selecting the project of interest on the CX/C++ projects view using the mouse, and then selecting the **Properties** menu item from the menu that appears by a right-click of the mouse.

This window contains several sections with parameters that can be set or modified by the developer in order to control the building process. The *Carbide Build Configuration* tab allows the setting build target specific options for the project. The *Paths and Symbols* tab allows the setting of paths related to the project, the customization of environment variables used for the building process, and the automatic applications that target Symbian OS v9.x phones. The *Carbide Project Settings* section allows the developer to select the components to build (i.e., MMP files) based on the contents of the imported bld.inf file.



Figure 9

Carbide.c++ allows the manipulation of a project's bld.inf file and its associated MMP files using a graphical editor which allows developers to customize the properties of their projects without the need to be familiar with the syntax and parameters of these files. These editors can be launched by opening these MMP files from the Symbian Project Manager or the C/C++ Project view. The MMP editor is shown in Figure 14.



Figure 10

The developer can then supply all the information required using the graphical editors or alternatively, click on the bld.inf or MMP file tabs on the bottom of the window in order to edit these files manually using a text editor.



Figure 11



Figure 12

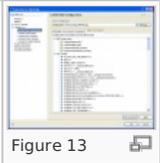


Figure 13



Figure 14

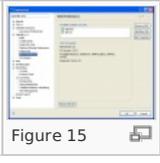


Figure 15



Figure 16



**Note:** Carbide.c++ always keeps these files synchronized with any modifications that the developer performs outside these editors, for example, the addition of source files to the project using the main Carbide.c++ user interface. Also, the manual editing of these files results in an automatic update of the information displayed in the graphical editors.

## Global preferences

Except for project-specific settings, Carbide.c++ also allows the developer to specify global options that affect the behavior of the IDE and all Symbian C++ projects. These settings are available on the *Preferences* window which can be launched by selecting the **Window > Preferences** menu item. As show in Figure 15, on the left hand side of this window the developer can select the group of options to modify, for example, general options, editor options, etc., while on the right hand side the options available for a group appear when the group is selected.

Of particular interest are the options available under Carbide.c++ tree item, which are specific to Symbian C++ development. The *SDK Preferences* tree item allows the developer to manage their installed Symbian Platform SDKs and to modify their properties. The *Build* item provides options for customizing the default build behavior of projects, for example, whether test code should be built.

Carbide.c++ provides keyboard shortcuts for commonly performed IDE tasks. The *Keys* option under the *General preferences* item provides the ability to modify these shortcuts. Either individual mappings can be altered, or a predefined Scheme can be chosen, such as the CodeWarrior or Visual Studio key mappings.

## Code navigation and editing

A significant amount of the developer's time using an IDE involves viewing and modifying source code. Carbide.c++ has many useful editor features, for example:

### Searching

**Search > Search...** provides the option to search for terms in a selection of files. The search can be restricted to certain file types, and to subsets of files defined by workspace, project or directory. Results can be viewed in either the *Search* or the *System Search* view.

### Keyboard shortcuts

Many of the editing and navigation features of Carbide.c++ have keyboard shortcuts associated with them. For example, Alt-Left and Alt-Right cycle forward and backwards through the files that are open in the editor, Ctrl + / comments and uncomments selected text, Ctrl+Alt+C compiles a project, and Ctrl+Alt+H opens a *Call* hierarchy for the identifier under the cursor. The available shortcuts for a perspective can be viewed in **Window > Preferences**.

### Type and Call hierarchies

Pressing F4 will open the *Type* hierarchy for the class name under the cursor. This will show the inheritance hierarchy and the functions defined at each level. As mentioned above, pressing Ctrl+Alt+H will open a *Call* hierarchy for the identifier under the cursor, and this will list all the locations in a project where the selected function is called, which can aid with refactoring.

### Code templates

Typing 'lit' and pressing Ctrl+Space will convert the string 'lit' to '\_lit("")'. There are many other code templates which are listed in **Window > Preferences > C/C++ > Editor > Templates**, for example templates for using the cleanup stack and control structures such as for and while loops, it is also possible to define new templates in this view.

### Local history

Right-clicking on a file in the *Project Explorer* and choosing **Team > Show Local History** displays a list of old versions of the file saved after each edit to the file. From this view the current version of the file can be compared to previous versions and changes reverted if necessary.

## Tasks

The *Tasks* view provides a way to manage development tasks. Tasks can either be added directly to the *Tasks* view or comments containing the word 'TO DO' can be added to source code and the comment's text will automatically be added to the *Tasks* view as a new task. These comment-generated tasks provide a quick way to access the related source code from the *Tasks* view.

## Split screen editing

By dragging and dropping an open file's editor tab to another part of the IDE, a second editor window can be created. Repeating this operation allows for creating as many editor windows as needed to enable viewing more than one source file at once, and viewing different parts of the same file simultaneously.

## Building and running an application

The Carbide.c++ IDE allows many different projects to be open simultaneously. As a result, it provides a facility to build all opened projects or selectively one or more opened projects. The steps to build a Symbian C++ project under Carbide.c++ are the following:

- select the project to build (click the project on the *Project Explorer* view)
- select the target type by using the **Project > Active Build Configuration** menu item
- build the project by selecting the **Project > Build Project** menu item

Carbide.c++ then builds the project by compiling, linking and optionally generating the SIS file for the target type selected by the developer. If required, a SIS file can be created at the end of the build by selecting the appropriate PKG file. This will generate and optionally sign the SIS file. Errors and warnings are displayed at the bottom of the IDE in the *Problems* view and the developer can click on the messages and automatically view the text that generated the error or the warning. The code that generated the error or warning will be underlined in the editor and marked with the appropriate error or warning icon. The *Console* view displays the commands that Carbide.c++ executes in order to build the project together with the raw error messages and warnings generated by the compiler, linker and the other building tools utilized.

After the building is completed and all errors have been fixed, the application can be launched on either the Symbian Platform emulator or a mobile phone, depending on the target type of the building process. If Carbide.c++ has all the information it needs it automatically generates a launch session configuration for the compiled project so that the developer can immediately launch the application by selecting the **Run > Run** menu item (or press Ctrl+F11) as well as a debug session configuration which can be started by selecting the **Run > Debug** menu item (or press F11). If Carbide.c++ does not find all the information it needs the *New Launch Configuration* wizard appears to walk you through any additional launch requirements before creating a launch configuration, then compiles, downloads, and launching it on the target device. These configurations can be modified and new configurations for running or debugging the application can be created using either the *Run* or the *Debug configuration* menu by selecting the **Run > Open Run Debug...** or the **Run > Open Debug Dialog...** menu item respectively.

As Figure 17 shows, for the case of the *Run* configuration window, the developer can create a run configuration which targets the Symbian Platform emulator or a mobile phone and specify various settings, such as command line options for the application, the USB or Bluetooth connection port for Carbide.c++ to send the SIS file to a phone for and on-targeted running/debugging, the location of the source files, and so on. Configurations for multiple projects and targets can be created and stored for use on successive Carbide.c++ sessions.

## Debugging sessions

A debug session is launched from one of a project's Debug Configurations as described in the previous section. A Debug Configuration allows the developer to customize how the application will be debugged, for example, the exceptions which Carbide.c++ should catch and whether an implicit breakpoint should be placed at an application's main function. Any time an application is executed under the debugger (either on the Symbian Platform emulator or a mobile phone), Carbide.c++ switches to the Debug perspective as shown in the next figure. Carbide.c++ provides support for all typical debugging features found in other IDEs. The threads composing the applications under execution are visible under the Debug view: the developer can select and examine a thread or process in this view to gather information about the thread's stack and local variables.

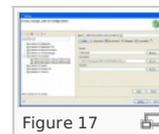


Figure 17

The Breakpoints tab allows the developer to create breakpoints and watchpoints. Breakpoints can be grouped and acted upon either individually or based on a grouping such as the file or project they were created for. When debugging using the emulator, watchpoints can be set to monitor the state of a particular variable, and this can be useful for debugging memory corruption bugs because Carbide.c++ will suspend threads which modify monitored variables. The value of global and local variables can be examined in the Variables view, which is capable of interpreting Symbian Platform-specific data, such as descriptors, and displaying their value. The values of registers and memory addresses can be viewed and modified in the Registers and Memory views respectively. These views can be used for analysing and modifying the state of an application.

Finally, the developer can step over or into functions, continue execution to the next breakpoint and finally restart or terminate the debugging session while viewing the source code and current line of code execution on a text editor.

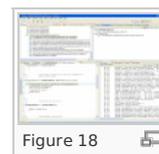


Figure 18

## New Features in Carbide.c++

There are a number of new features in Carbide.c++ not already mentioned. These include:

### 2.0.4

- **Hover Help** - hover your pointer over a recognized Symbian symbol in an editor and see the Symbian API Reference information appear to assist you.
- **Carbide.c++ News Reader** - introducing the *Carbide.c++ News* view which provides the latest Carbide, Symbian, and S60 news as well as news on tools updates and screencasts for Carbide development.

### 2.0.0

- **Eclipse 3.4.1** — The Eclipse 3.4 project (Ganymede) provides the foundation, or integration platform, on which Carbide.c++ is built.
- **CDT 5.0.1** — The final version of CDT included as part of the Ganymede release.

- **SDK Migration support in CodeScanner** — CodeScanner now includes support for S60 SDK rules that provide helpful migration information within any C/C++ editor view and provide links to additional information within the SDK documentation or web. Very helpful during development to keep you code up-to-date with the latest SDK revisions.
- **Qt SDK for S60 support** — Carbide now supports the importing and creation of Qt projects using the Qt SDK for S60.
- **Executables view improvements** — The *Executables view* now supports the loading of some additional DLL cases to ensure that the DLLs you want to debug are available during a debug session. In addition, a Remove button has been added to the view's toolbar that makes it easier for you to remove any executables from the list.
- **More indexer improvements** — Speed improvements for code completion, syntax coloring, and much more.
- **Remote Connections view** — A visual list of remote connections and their state for managing multiple and diverse device connections.
- **Partial upgrade SIS file support** — Carbide now supports the generation of partial upgrade SIS/SISX files. This increases debugger turnaround by only generating new files for changes and uploading these smaller executables to the device.
- **More Program Counter support** — Added a *Move to Line* command for the debugger that enables you to move the PC to a new line while debugging without executing any intermediate lines or resuming execution.
- Multiple **TRK** improvements including:
  - Both App and System TRK try to disable mobile crash debugging
  - TRK now looks for the new mobile crash agent (ms\_useragent)
  - Added additional error checking when communicating over USB to improve stability
  - Fixed some random TRK deadlocks when reading registers or when thread panics occur
  - **Option > Exit** has been replaced by **Option > Back** to reflect the change made to exiting TRK
  - Changes made to use UIDs in place of component names to identify executables for improved identification
  - Reset TRK priority to high to avoid problems debugging complex process intensive applications
  - Improved breakpoint location in DLLs when debugging multiple processes
  - TRK no longer targets processes with UID 0 automatically. Users can still attach to the process or use a launch configuration.
  - TRK application name and UID changed to blocking of TRK SIS files assigned with the old UID.
  - TRK now always uses the kernel APIs to read/write memory to improve security, no more direct reads/writes.
  - Application TRK no longer allows write operations to the following registers: CPSR, SP, and LR

### 1.3.1

- **On-Device Setup dialog** — connects, verifies and updates Carbide software services like the TRK debug agent on a device to ensure you are always using the latest version in your development efforts. Having trouble connecting with TRK, give this a try.
- **New Bug Report wizard** — directly submit bugs and enhancement requests to Carbide Bugzilla from within Carbide using the **Help > New Bug Report** menu option. Enter the name and password into the Bug Reporter preference panel and quick bug reports are easy to submit.
- **Launch configuration panels UI unification and improvements** — a host of minor UI improvements are being made in the launch configuration panes to provide a more coherent and reproducible experience across all launch configuration dialogs. For example, the concept of a "main executable" had been replaced with the Executables tab where a list of executables targeted for debugging were shown. This UI change brings the launch configurations into sync with that change by removing the Main Executable from the *Main* tab. The Main tab will only show a process to launch and not executables targeted for debugging. Another example, the *Arguments* field used by the run-mode TRK configurations has been moved to the *Arguments* tab to match the Emulator configuration.
- Many other bug fixes to improve stability and usability on Carbide

### 1.3.0

- Optional build system features which can help speed up building projects. These options are accessible through the **Preferences > Carbide.c++ > Build** pane and allow the developer to direct Carbide.c++ to manage build dependencies natively and to run builds concurrently on multi-core machines.
- A System Search feature is available through the **Search > System...** option. This allows searching in arbitrary directories on a machine and is not tied to only searching files which are part of the workspace.
- Project source files can be preprocessed by right-clicking on a file in the *Symbian Project Navigator* and choosing *Preprocess*. The C-preprocessor will be run over the source file and the output displayed in a *Console* view.
- When creating a new workspace, the current workspace's settings can be copied to the new workspace. The option is available after selecting **File > Switch Workspace > Other...** and then choosing which workspace aspects to copy.
- The Platform Security view can provide an analysis of the capabilities that an MMP file will need. By right-clicking on an MMP file and choosing Run Capability Scanner on Project MMP, the capabilities that the binary file will require are summarised in the Platform Security view.
- As the Carbide.c++ ecosystem grows, there are an increasing number of plug-ins available which extend the functionality of Carbide.c++. Some of these plug-ins are Symbian-specific and are developed by Symbian or Nokia, while others are more generic to Eclipse and come from other tools vendors.

## Information and Help

In this booklet we provided some basic information about Carbide.c++, including how to import, build, debug and execute Symbian C++ projects. More information on all features of Carbide.c++ can be found within the IDE, which can be accessed by the **Help > Help Contents** menu item.

Help is also available for the Eclipse IDE, the Carbide.c++ specific features on top of the basic Eclipse IDE, as well as for the installed Symbian SDKs. You can find additional useful information in the [Carbide.c++ category](#) of this wiki.



© 2010 Symbian Foundation Limited. This document is licensed under the [Creative Commons Attribution-Share Alike 2.0](http://creativecommons.org/licenses/by-sa/2.0/legalcode) license. See <http://creativecommons.org/licenses/by-sa/2.0/legalcode> for the full terms of the license.

Note that this content was originally hosted on the Symbian Foundation developer wiki.

