

Introduction to Carbide.c++

Panos Asproulis

Version: 1.0

1	INTRODUCTION	1
2	THE ECLIPSE PLATFORM	2
3	THE ECLIPSE ARCHITECTURE.....	3
4	THE ECLIPSE IDE	4
5	THE CARBIDE.C++ IDE	9
6	SOURCES OF INFORMATION	15

1 Introduction

This article provides an introduction to the new *Integrated Development Environment* (IDE) for the development of Symbian C++ applications called Carbide.c++. It mainly describes the currently released free version of this IDE, which is called Carbide.c++ Express and can be downloaded from the [Forum Nokia](#) and the [UIQ Developer](#) web sites. Other commercial versions with more features are scheduled for release in the near future and therefore more articles will follow to describe these versions when they are officially released. Any description of Carbide.c++ cannot be complete without a description of the Eclipse Platform which is used as the basis for the Carbide.c++ line of products and therefore introductory information on this platform is the starting point for this article.

2 The Eclipse Platform

The Eclipse Platform was launched in November 2001 by IBM and seven other companies as an open source project targeting the creation of a multi-platform IDE capable of addressing the needs of the Java developers, whilst at the same time offering a free alternative to commercial products. In fact, IBM donated a \$40 million internal development effort to the international open source community, a move which was considered as rather bold at the time.

Although Eclipse started as a mainly Java development environment, it soon became a far more complicated and flexible piece of software, eventually becoming an integration platform for tools and client applications. Maintained by the non-profit *Eclipse Foundation* organisation (Eclipse.org) and its member companies, which includes Symbian, 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 and Carbide.c++ is one of them.

The real goal for Eclipse is an environment that makes the integration of various types of tools easy. Eclipse fulfils this requirement by utilising an architecture that implements a large set of reusable frameworks which can be utilised by any tool. It contains a framework which implements a consistent user interface, a framework for organising and accessing programming artefacts from the file system, frameworks for text editors, application builders, debugging sessions, team programming, version control etc. Still, the most important aspect of Eclipse's architecture is the fact that new code can be easily integrated to 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.

All Eclipse code is a pluggable unit to a core microkernel which is responsible for detecting and enabling these units, as shown in Figure 2.1. In that respect, Eclipse can be compared to an operating system with a small kernel and a large number of modules which can be customised in order to produce a specific system with the required functionality.

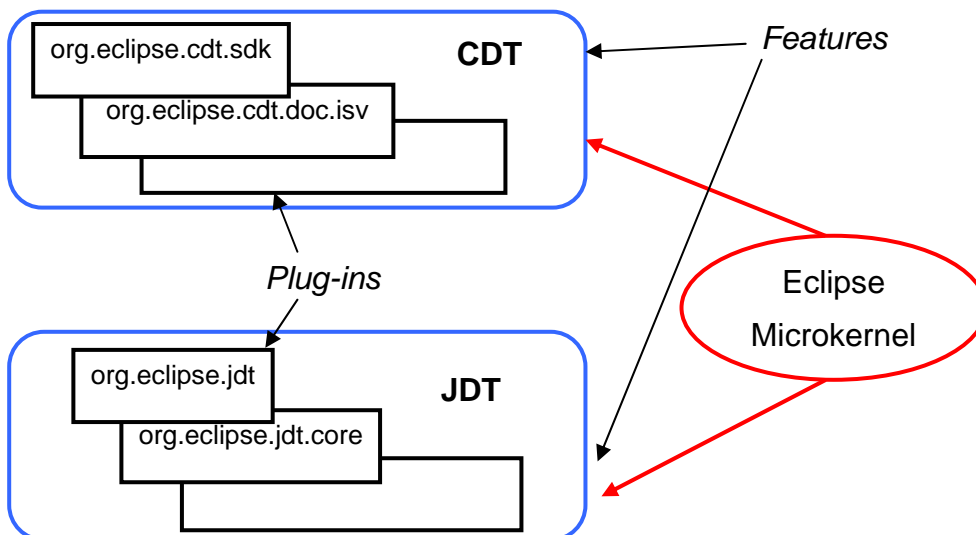


Figure 2.1 – Eclipse Plug-ins and Features

These units are called *Eclipse plug-ins* and they are composed of a set of Java source files that implement some functionality. For example, in Figure 2.1 plug-in `org.eclipse.jdt` implements some functionality to allow the development of Java applications on Eclipse. Each plug-in contains a text-

based declaration file which is called a *bundle manifest* (`manifest.mf`) with information that Eclipse requires in order to activate the plug-in. Eclipse plug-ins which implement some major functionality, e.g. C++ development, can be grouped together and represented by an *Eclipse feature*. In contrast to a plug-in, an Eclipse feature does not contain any source code but it is simply a mechanism for Eclipse to manage, organise and update a major functionality that a set of plug-ins implement or other Eclipse features. It is therefore a logical container for plug-ins and it is capable of nesting other related features. For example, in Figure 2.1 the collection of `org.eclipse.jdt.*` plug-ins allow Eclipse to support the development of Java applications and they all form the *Java Development Toolkit* (JDT) feature.

The Eclipse plug-ins and features are normally stored in two separate directories under the main Eclipse installation directory, but it is also possible for them to exist in any location on the filesystem as long as this is explicitly specified by the user on the Eclipse configuration management panel.

3 The Eclipse Architecture

A simplified graphical representation of the Eclipse architecture is displayed in Figure 3.1 below.

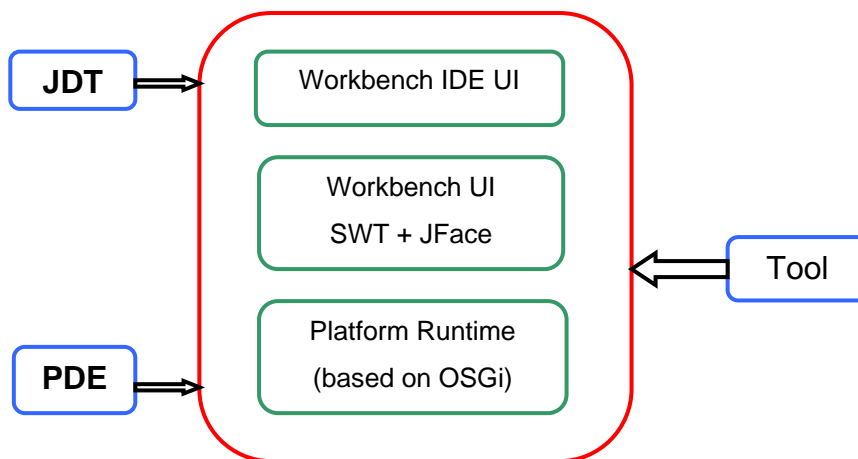


Figure 3.1 – The Eclipse Architecture

Eclipse is essentially the composite of three main units:

- The **Platform Runtime**, which is an [OSGi](#)-compliant microkernel responsible for managing the Eclipse plug-ins. This component handles the Eclipse start-up procedure, discovers all the locally installed plug-ins, reads their manifest files and finally matches the declared extensions with the available extension points. The result is a registry in memory with this information that is available to all plug-ins. Carbide.c++ is based on version 3.x of Eclipse, which introduced the OSGi component-based *Platform Runtime* and therefore is capable of loading plug-ins dynamically without having to restart the product for new plug-ins and extensions to be recognised.
- The **Workbench UI, SWT and JFace**, which represent all the graphics and user interface frameworks implemented in Eclipse which are available for all Eclipse-based applications. The *Workbench UI* contains the editors, views and perspectives that make the user interface personality of Eclipse. The *Standard Widget Toolkit* (SWT) is a generic, portable, low-level graphics and widget set built upon the native controls provided by the host operating system (e.g. buttons, menus etc). If a particular widget is not available in the native environment SWT provides an emulated equivalent. Finally, *JFace* is a model-based user interface framework for high level UI components.

- The **Workbench IDE UI**, which is the set of frameworks that implement the integrated development environment user interface functionality. It contains the various document and source file editors, compare and search facilities, the *Eclipse Workspace* and its resources, support for version control systems etc.

As shown in Figure 3.1, other components can be added to this system to provide further functionality so that specific products can be created. The **Java Development Toolkit** (JDK) is a set of Eclipse plug-ins that support Java development on the *Eclipse IDE*. When combined with the **Plug-in Development Environment** (PDE) component they result in the *Eclipse Software Development Kit* (SDK) which can be used both for the development of Java applications as well as Eclipse plug-ins. Other tools can be similarly added.

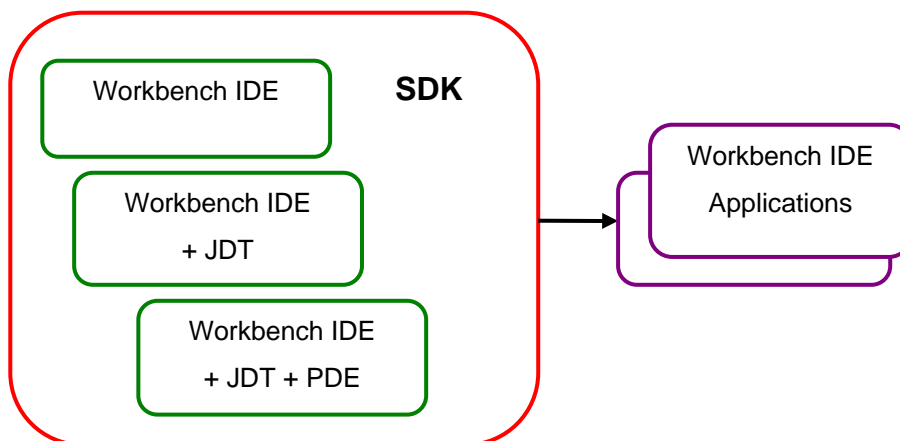


Figure 3.2 – Eclipse-based IDE Applications

Figure 3.2 shows that Eclipse-based IDEs can be created by using the Eclipse SDK and the frameworks it contains. Carbide.c++ is such a product, developed using the Eclipse SDK and based on the Workbench IDE.

4 The Eclipse IDE

The Eclipse IDE can be downloaded from the Eclipse.org web site for a large variety of platforms. The IDE does not support the development of Eclipse plug-ins; for that purpose the Eclipse SDK should be downloaded instead. After launching Eclipse one is presented with the Eclipse Workbench IDE, which is shown in Figure 4.1 for a Windows platform. Note that on each platform Eclipse will use the host operating system's look and feel because its widget engine is using SWT, which as mentioned in Chapter 3, is utilising the native widgets. Therefore, the Eclipse IDE will look different when executed on another operating system, e.g. Linux, compared to what is shown in Figure 4.1, although the menus and views will be in the same location and have the same functionality.

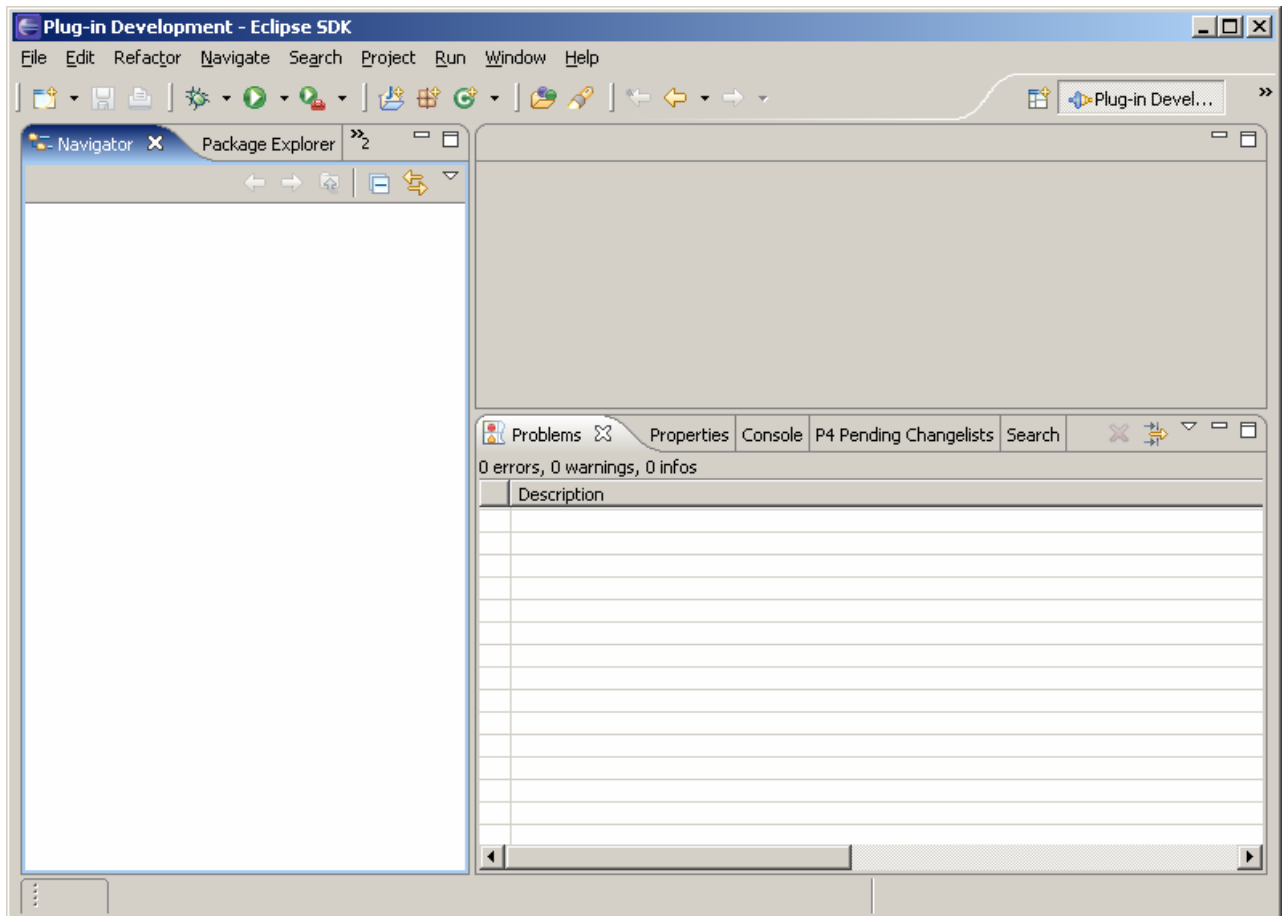


Figure 4.1 – The Eclipse Workbench IDE

All the windows within the IDE can be customised and their positions changed. The Eclipse IDE/SDK is ready to use for the development of Java applications. When a Java project is created, the Eclipse IDE, typically configured, displays the contents of the project on the left-hand side of the IDE on both the *Navigator* and the *Package Explorer* views. The *Navigator* view displays the files and structure of the project directory while the *Package Explorer* displays the Java packages and classes comprising the project. The source files are displayed on the Java text editor, as shown in Figure 4.2 below, which is language-sensitive and uses different colours and/or fonts to mark the various Java language elements. Eclipse provides support for the implementation of language-sensitive editors, a facility used by Carbide.c++ for displaying Symbian C++ source files. Finally, the Eclipse source editors provide support for code and comment folding in order to allow the developer to adjust the detail of the displayed information. A developer who simply wants to see the structure of the classes implemented in a project without the implementation details can choose to fold all member functions and quickly browse the source files.

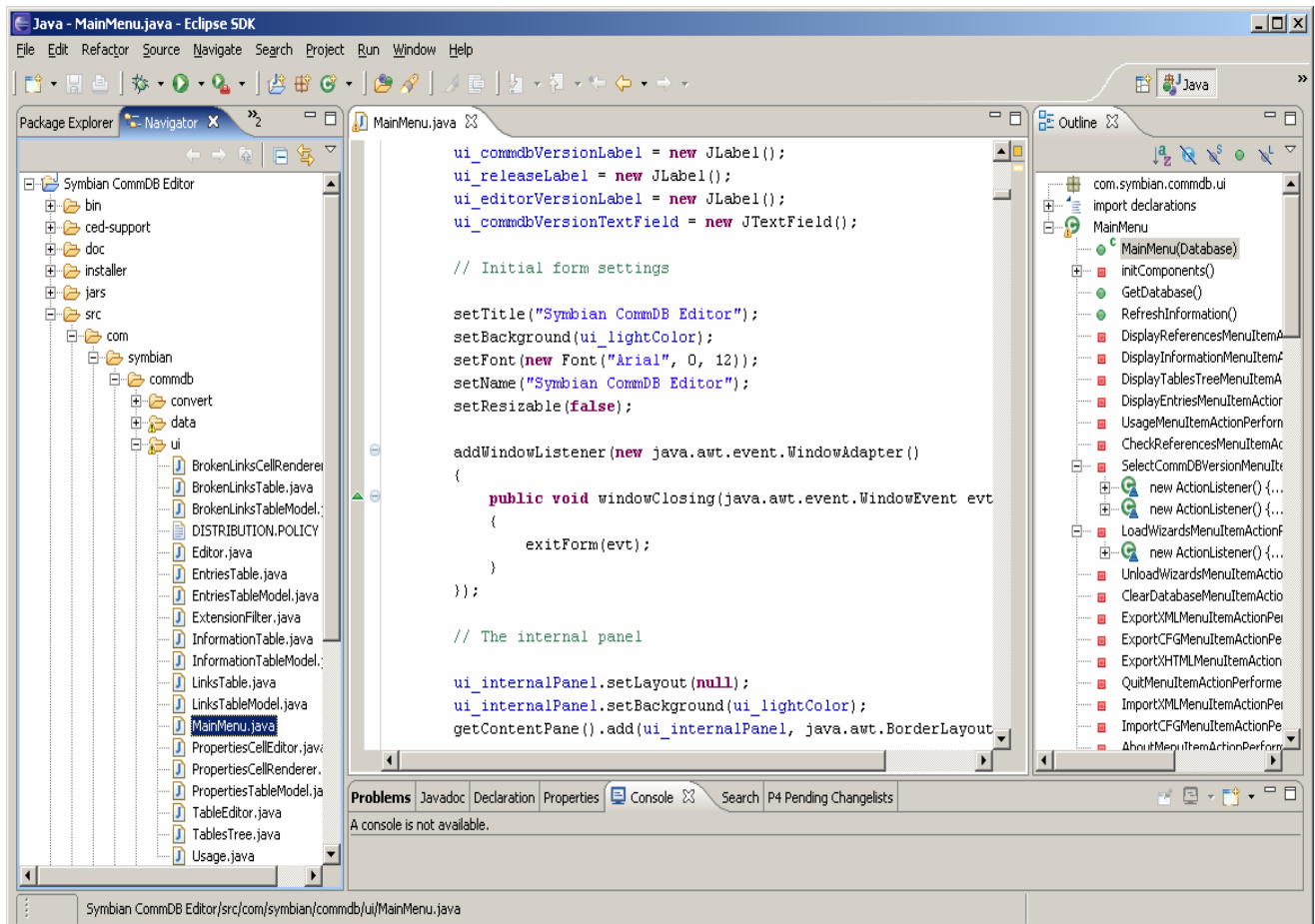


Figure 4.2 – The Eclipse IDE with a Java Project

One issue that most developers encounter with the Eclipse IDE is that it does not try to imitate the look and feel of any other popular development environment, e.g. Microsoft's Visual Studio. Those intimately familiar with a particular IDE need to spend some time to familiarise themselves with the Eclipse environment as well as understand its architecture and philosophy. It is important to realise that Eclipse is designed using the principles of *Object Oriented Programming (OOP)* and the extensive reuse and extension of frameworks. In essence, all visual elements on the Eclipse IDE are objects of some kind which can be interrogated to provide useful information. For example, a right-click on a file as shown in Figure 4.2 will launch a menu with actions and information on the selected file. The same principle applies to all elements in the views of Eclipse, including the top directory folder which represents the currently opened project. Available actions vary according to the type of the selected element, e.g. a file can be edited, deleted etc., while the top directory folder can be built since it represents the project itself. The properties of a project, e.g. compiler and linker options, are not available in some top level menu but on the element that represents the project itself. A right-click mouse action on the project will reveal a menu with a properties menu item that, once selected, will display a window with all the options related to building and managing the selected project.

Configuring the various Eclipse global settings as well as settings particular to a feature or plug-in is managed by the *Preferences Panel* shown in Figure 4.3. On Windows systems this can be launched by selecting the *Windows->Preferences* menu item.

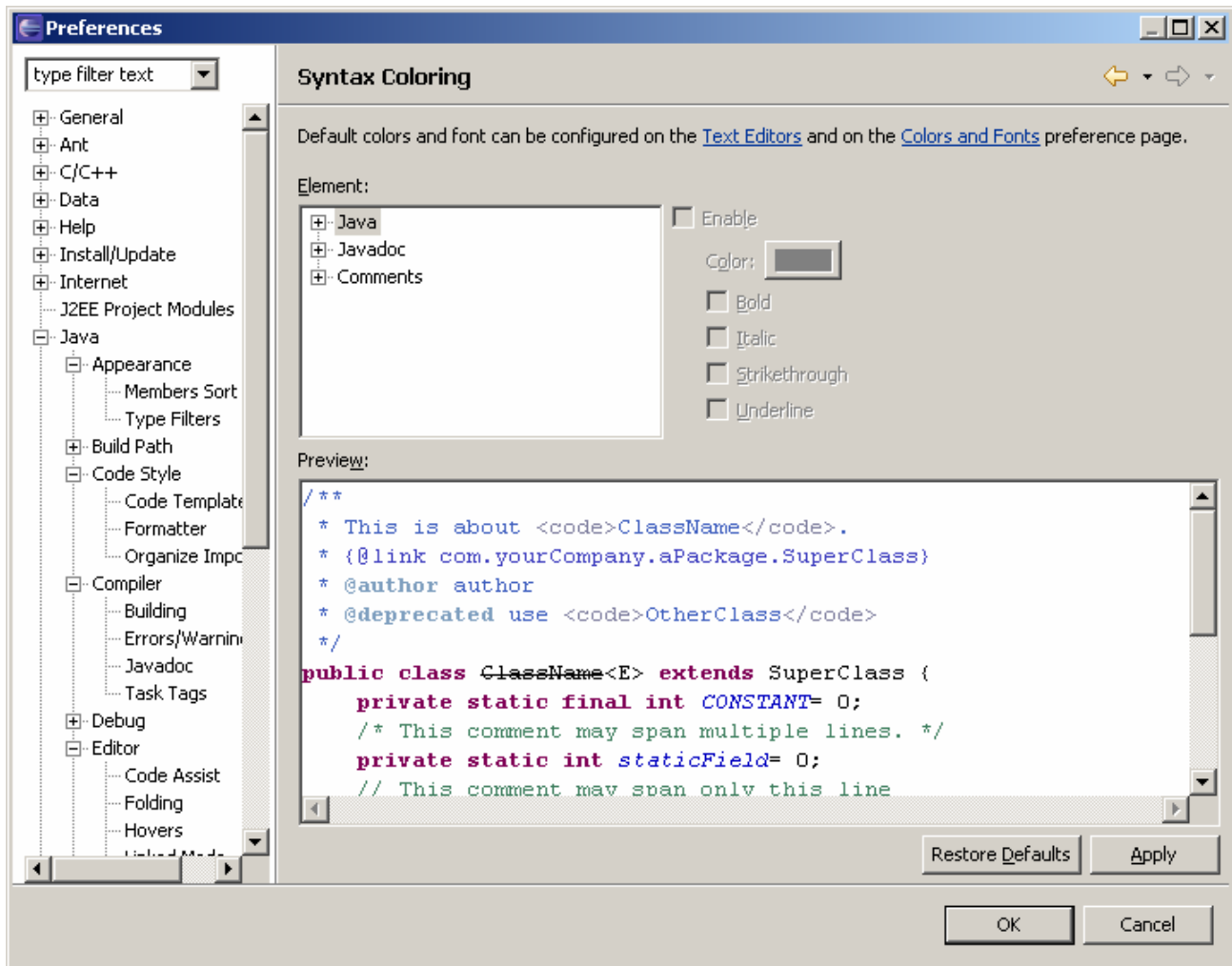


Figure 4.2 – The Eclipse Preferences Panel

As explained in Chapter 3, any Eclipse-based product is composed of a number of plug-ins, usually grouped in Eclipse features. Eclipse provides a way for the user to see all the currently installed and activated components by selecting the *Help->About Eclipse* menu item. By clicking the *Feature Details* button, the user is presented with a list of all the installed features. An example of this information for the case of an Eclipse SDK-based installation is shown in Figure 4.3. Clicking on the *Plug-in Details* button produces a similar list with all the installed plug-ins. Alternatively, one can select a particular feature in the Eclipse features view and then click the *Plug-in Details* button (shown in Figure 4.3) in order to see only those plug-ins that form the selected feature. For example, selecting the *C/C++ Development Toolkit* (CDT) feature displayed in Figure 4.3 produces the view shown in Figure 4.4, i.e. all the plug-ins which allow Eclipse to manage and build C/C++ applications.

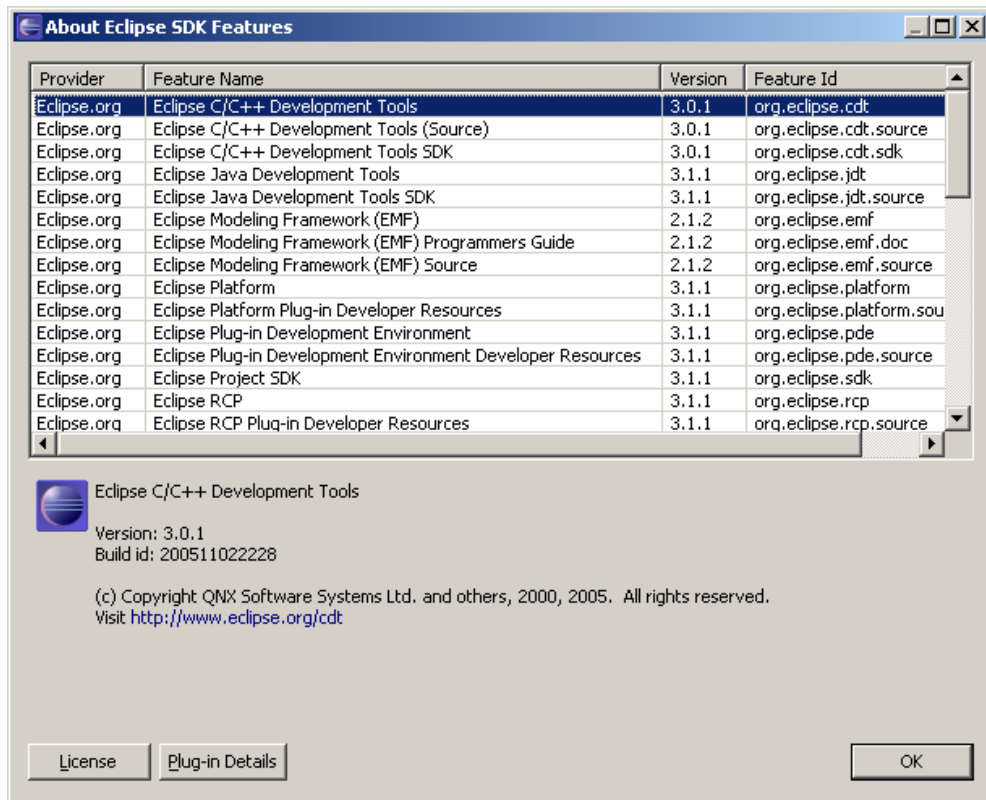


Figure 4.3 – Installed Eclipse Features View

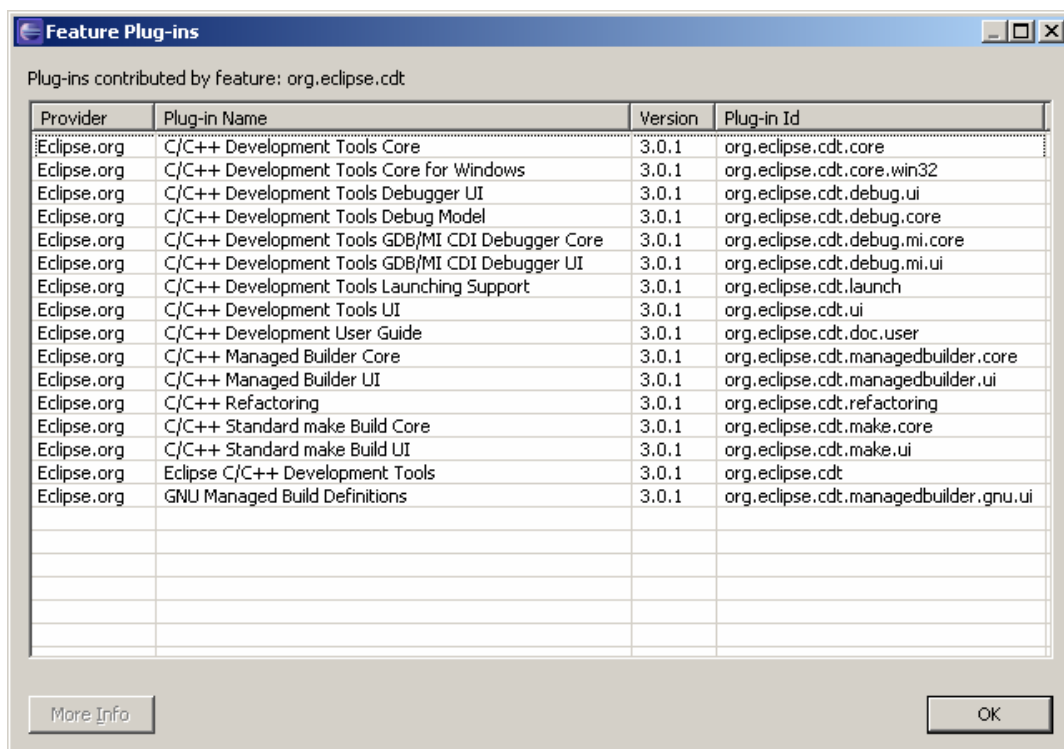


Figure 4.4 – The Eclipse Plug-ins of the C/C++ Development Toolkit (CDT) Feature

The Eclipse IDE provides support for building the source files forming a project using the building tools installed on the host computer. So, Java applications can be compiled by installing a Java Development Kit (JDK), while C/C++ applications require the installation of a compiler supported by the C/C++ Development Toolkit on the host platform e.g. the GCC compiler on UNIX systems. Certainly, the Eclipse SDK allows the development of any type of builder for applications and in fact one can easily find Eclipse plug-ins in Internet that provide support for various popular programming languages and compiler/debugger systems. The most popular web sites with all types of plug-ins for Eclipse are the [Eclipse Plugin Central](#) and [Eclipse Plugins](#).

5 The Carbide.c++ IDE

The Carbide.c++ IDE is based on the Eclipse IDE Version 3 with additional plug-ins which make Eclipse understand how to handle Symbian C++ source files as well as build Symbian projects. These additional plug-ins can be split in the following categories:

- The *C/C++ Development Toolkit* (CDT), which is a set of plug-ins that make Eclipse capable of managing and building C/C++ projects. These plug-ins implement the C/C++ Project view, the C/C++ source editor, the Make and Auto-Make facilities for building C/C++ applications using the host computer's C/C++ compiler and a Make utility etc. These plug-ins are essential for any Eclipse-based IDE product that supports C/C++ programming.
- *Symbian Plug-ins*, which have been developed by Symbian and enable Eclipse to handle Symbian C++ concepts, whilst providing support for Symbian OS SDKs and the Symbian Toolchain.
- *Nokia Plug-ins*, which have been developed by Nokia and provide further support for Symbian OS SDKs. They implement the Metrowerks C++ compiler and debugger for Symbian OS and add some advanced features for the commercial line of the Carbide.c++ product line.

Obviously, since Carbide.c++ is essentially the Eclipse IDE, it can be further customised by the developer with the installation of additional commercial or free plug-ins, such as a version control system, a UML modelling system etc.

The installation of Carbide.c++ is simple as it comes with its own installer. After launching the product, the user is displayed with an IDE which is mostly identical to the Eclipse IDE and it is shown in Figure 5.1. Since Carbide.c++ is designed to handle Symbian C++ applications, the left-hand side of the IDE now displays the *C/C++ Project* view which is designed to display the Symbian C++ source, header and resource files that compose a typical Symbian C++ project. On the right-hand side the *Build Configurations* view is available, which is designed to display the target platform and build configuration that the user has selected for an opened Symbian C++ project.

Feature Plug-ins

Plug-ins contributed by feature: com.nokia.carbide.cpp

Provider	Plug-in Name	Version	Plug-in Id
Freescal	C/C++ Development Tools CodeWarrior Debugger Core	1.0.0	com.freescal.cdt.debug.cw.core
Freescal	C/C++ Development Tools CodeWarrior Debugger UI	1.0.0	com.freescal.cdt.debug.cw.core.ui
Freescal	Freescal Swt Extension Plug-in	1.0.0	com.freescal.swt
Nokia	Capabilities Plug-in	1.0.0	com.nokia.cdt.debug.capabilities
Nokia	Carbide.c++	1.0.0	com.nokia.carbide.cpp
Nokia	Carbide Help Plug-in	1.0.0	com.nokia.cdt.debug.help
Nokia	Carbide Support Plug-in	1.0.0	com.nokia.carbide.cpp.support
Nokia	C Compiler Help Plug-in	1.0.0	com.nokia.carbide.compiler.cpp.help
Nokia	Launch Plug-in	1.0.0	com.nokia.cdt.debug.launch
Nokia	Nokia Common Templates Plug-in	1.0.0	com.nokia.cdt.templates
Nokia	SDK 560_12 WINSCW Build Plug-in	1.0.0	com.nokia.sdk.560_12.build.winscw
Nokia	SDK 560_20 WINSCW Build Plug-in	1.0.0	com.nokia.sdk.560_20.build.winscw
Nokia	SDK 560_21 WINSCW Build Plug-in	1.0.0	com.nokia.sdk.560_21.build.winscw
Nokia	SDK 560_26 WINSCW Build Plug-in	1.0.0	com.nokia.sdk.560_26.build.winscw
Nokia	SDK 560_28 WINSCW Build Plug-in	1.0.0	com.nokia.sdk.560_28.build.winscw
Nokia	SDK 560_30 GCCE Build Plug-in	1.0.0	com.nokia.sdk.560_30.build.gcce
Nokia	SDK 560_30 WINSCW Build Plug-in	1.0.0	com.nokia.sdk.560_30.build.winscw
Nokia	SDK 560 1.2 GCC98 Build Plug-in	1.0.0	com.nokia.sdk.560_12.build.gcc98
Nokia	SDK 560 2.0 GCC98 Build Plug-in	1.0.0	com.nokia.sdk.560_20.build.gcc98
Nokia	SDK 560 2.1 GCC98 Build Plug-in	1.0.0	com.nokia.sdk.560_21.build.gcc98
Nokia	SDK 560 2.6 GCC98 Build Plug-in	1.0.0	com.nokia.sdk.560_26.build.gcc98
Nokia	SDK 560 2.8 GCC98 Build Plug-in	1.0.0	com.nokia.sdk.560_28.build.gcc98
Nokia	SDK S80 DP2 GCC98 Build Plug-in	1.0.0	com.nokia.sdk.S80_DP2.build.gcc98
Nokia	SDK Series80 DP2 WINSCW Build Plug-in	1.0.0	com.nokia.sdk.S80_DP2.build.winscw
Nokia	SDK UIQ_21 WINSCW Build Plug-in	1.0.0	com.nokia.sdk.UIQ_21.build.winscw
Nokia	SDK UIQ_30 GCCE Build Plug-in	1.0.0	com.nokia.sdk.UIQ_30.build.gcce
Nokia	SDK UIQ_30 WINSCW Build Plug-in	1.0.0	com.nokia.sdk.UIQ_30.build.winscw
Nokia	SDK UIQ 2.1 GCC98 Build Plug-in	1.0.0	com.nokia.sdk.UIQ_21.build.gcc98
Nokia	Symbian Plug-in	1.0.0	com.nokia.cdt.debug.cw.symbian
Nokia	Wizards Plug-in	1.0.0	com.nokia.cdt.debug.wizards
Symbian	Abstract Build Support Plug-in	1.0.0	com.symbian.cdt.build
Symbian	Project Plug-in	1.0.0	com.symbian.cdt.project
Symbian	SDK 7.0a GCC98 Build Plug-in	1.0.0	com.symbian.cdt.sdk.70a.build.gcc98
Symbian	SDK 7.0a WINSCW Build Plug-in	1.0.0	com.symbian.cdt.sdk.70a.build.winscw
Symbian	SDK 7.0s GCC98 Build Plug-in	1.0.0	com.symbian.cdt.sdk.70s.build.gcc98
Symbian	SDK 7.0s WINSCW Build Plug-in	1.0.0	com.symbian.cdt.sdk.70s.build.winscw
Symbian	SDK 8.0a GCC98 Build Plug-in	1.0.0	com.symbian.cdt.sdk.80a.build.gcc98
Symbian	SDK 8.0a WINSCW Build Plug-in	1.0.0	com.symbian.cdt.sdk.80a.build.winscw
Symbian	SDK 8.1a GCC98 Build Plug-in	1.0.0	com.symbian.cdt.sdk.81a.build.gcc98
Symbian	SDK 8.1a WINSCW Build Plug-in	1.0.0	com.symbian.cdt.sdk.81a.build.winscw
Symbian	SDK 9.1 GCCE Build Plug-in	1.0.0	com.symbian.cdt.sdk.91.build.gcce
Symbian	SDK 9.1 WINSCW Build Plug-in	1.0.0	com.symbian.cdt.sdk.91.build.winscw
Symbian	Symbian CDT Core Plug-in	1.0.0	com.symbian.cdt.core
Symbian	Symbian SDK Support Plug-in	1.0.0	com.symbian.cdt.sdk
Symbian	Template Engine Plug-in	1.0.0	com.symbian.cdt.templateengine

More Info OK

Figure 5.2 – The Plug-ins that compose the Carbide.c++ Express IDE

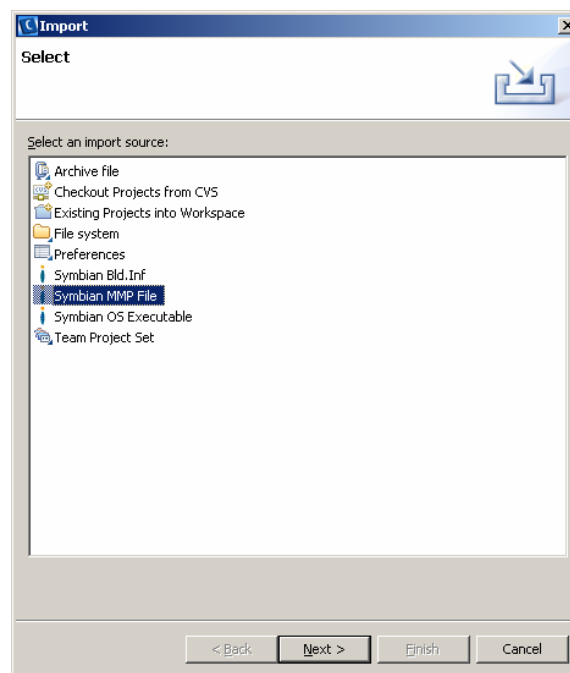


Figure 5.3 – The Import Facility of Carbide.c++ Express

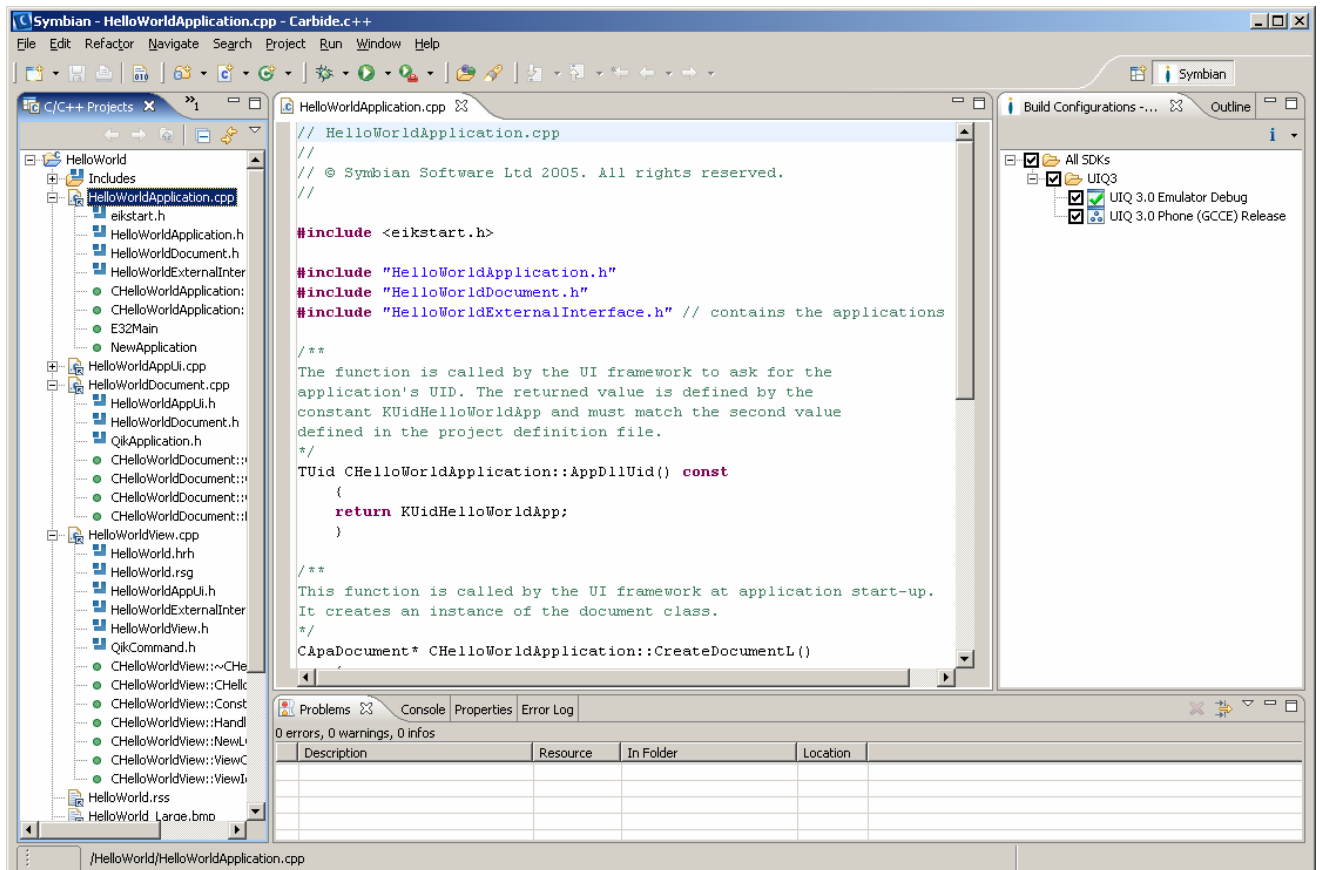


Figure 5.4 – The HelloWorld Application on Carbide.c++ Express

Note that the Build Configurations view in Figure 5.4 now displays the available SDKs which are appropriate for the selected project, in this case UIQ 3.0, as well as the supported build configurations, in this case a build for the Symbian OS Emulator and a build for an actual phone with UIQ 3.0, compiled using the free GCC-E compiler. In this example, both build configurations are selected, which means that when Carbide.c++ is asked to build the project it will create binaries for both targets. Note that Carbide.c++ supports both the free [GCC-E](#) and the commercial [ARM RVCT](#) compilers for building binaries for phones and the presence of these compilers on the host computer is automatically detected. Carbide.c++ extracts information regarding the installed Symbian OS SDKs on the host platform by parsing the `devices.xml` file which in a typical installation is stored in the `C:\Program Files\Common Files\Symbian\` directory. Using this information, it allows the developer to select one or multiple SDKs when he either imports or creates from scratch a Symbian C++ project.

The properties of an opened Symbian C++ project in Carbide.c++ can be viewed by selecting the project on the *C/C++ Project* view and then performing a right-click. This will bring up a submenu with a *Properties* menu item. When this item is selected, the *Properties* window appears on the screen; Figure 5.5 shows this for the *HelloWorld* application. This window contains extensive information regarding the project and the developer can edit various settings, e.g. the compiler options, linker options as well as the UIDs and the Platform Security capabilities of the application or library under development. It is important to note that if a developer uses the Symbian command-line toolchain to build a Symbian C++ project, he has no access to some of the options that are available in the *Properties* window of Carbide.c++, e.g. the compiler options. Unless he uses the default values provided by Carbide.c++, the binaries generated by the command-line toolchain will be different to those generated by Carbide.c++.

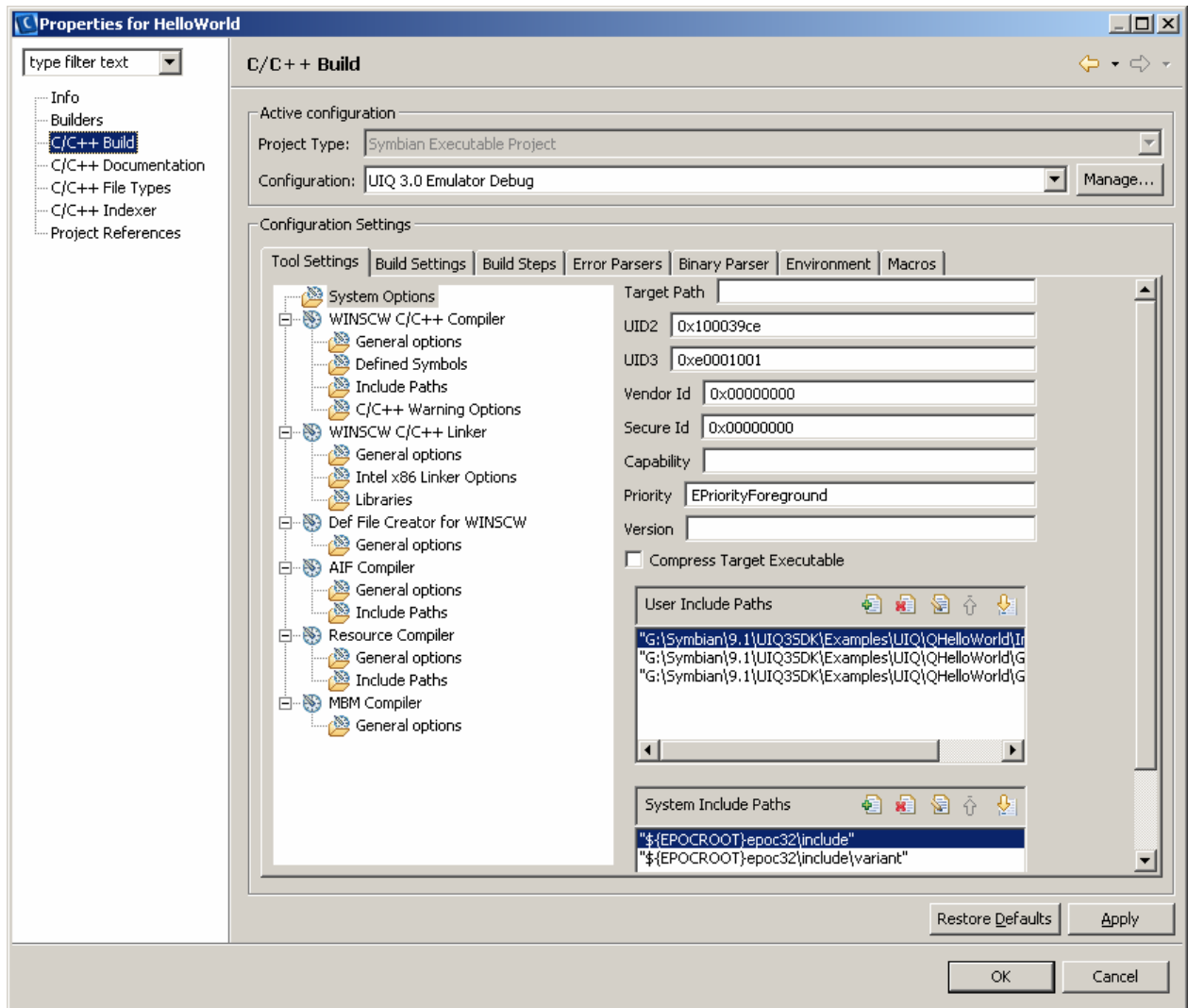


Figure 5.5 – Properties for the HelloWorld Project

Carbide.c++ also includes extensive help information regarding both the standard Eclipse IDE and the additional capabilities added, including the compiler and debugger which are installed with the Carbide.c++ IDE by the installer. The help information can be launched by selecting the *Help->Help Contents* menu item and this is shown in Figure 5.6.

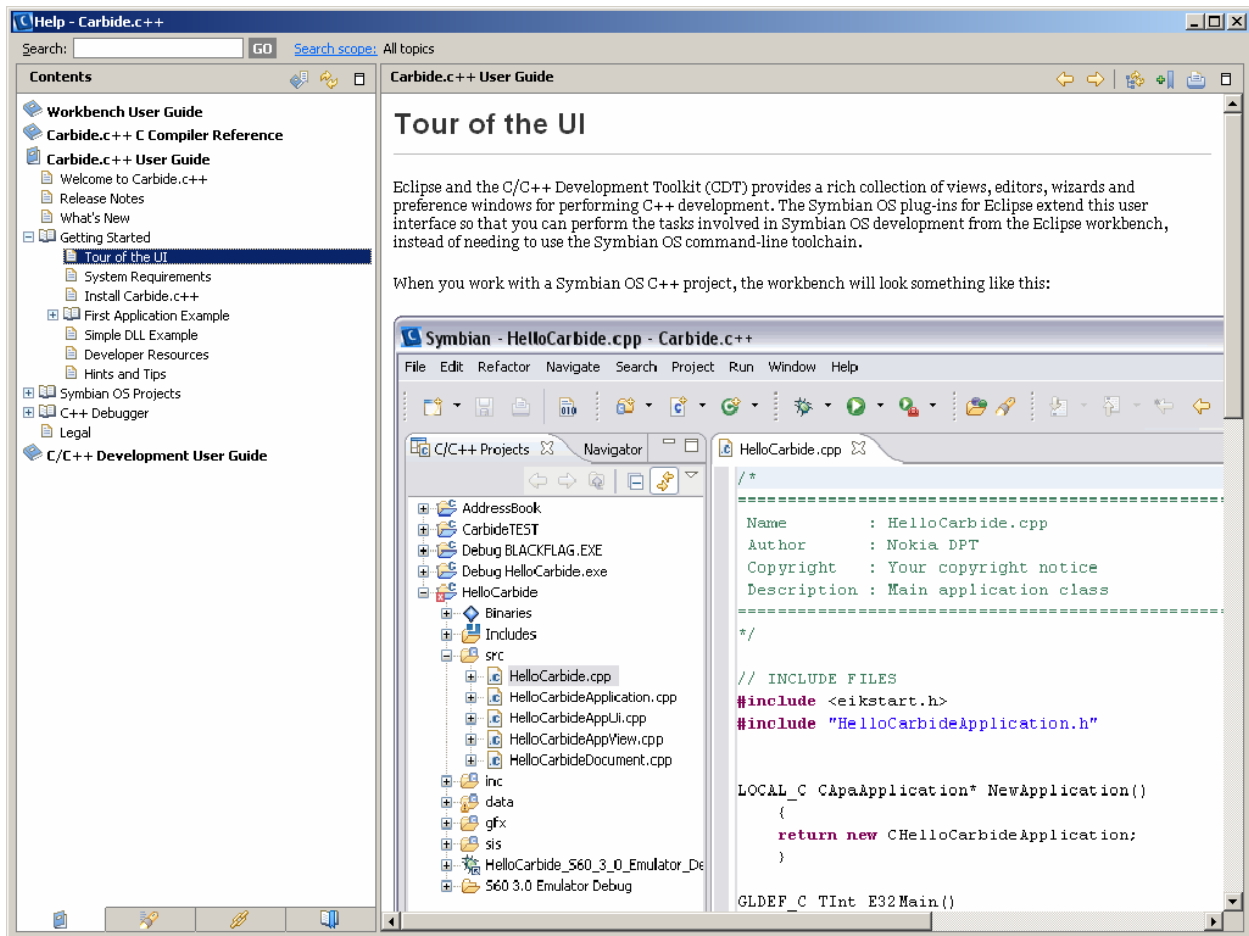


Figure 5.6 – Carbide.c++ Help Information

Eclipse provides a facility to update the installed plug-ins by obtaining new versions from the Internet. This is also utilised by Carbide.c++ and it can be performed by selecting the *Help->Software Updates->Find and Install* menu item, as shown in Figure 5.7. The user can select which update sites to visit and then which particular components to update, if updates are available. The same facility can be used in order to install additional plug-ins, since most Eclipse plug-in developers provide a URL which can be provided to the Eclipse update system for downloading the required components.

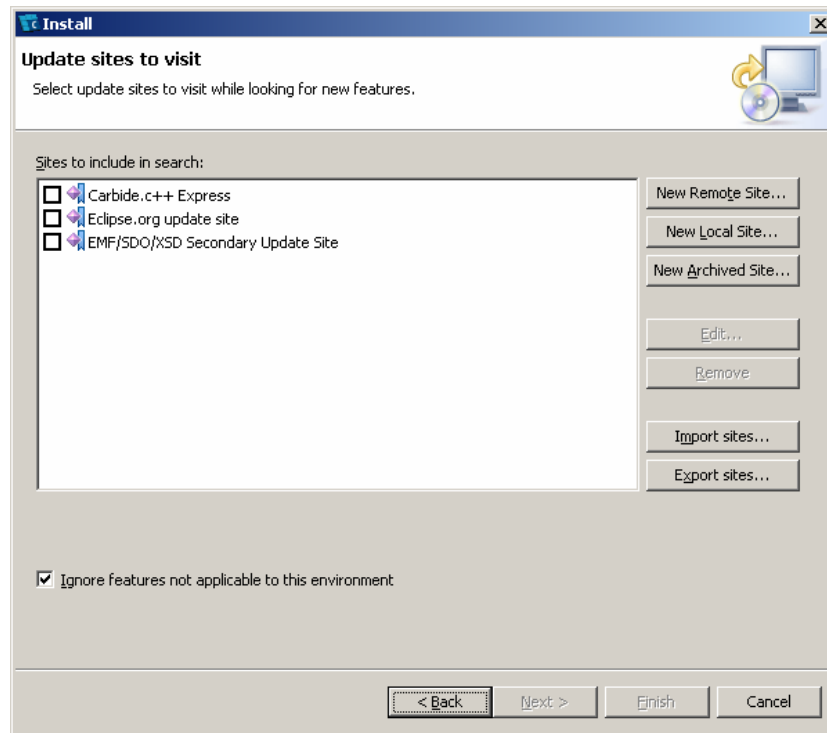


Figure 5.7 – Carbide.c++ Update Facility

6 Sources of Information

Those interested to learn more about the architecture of Eclipse, learn to use the Eclipse IDE as well as develop plug-ins for Eclipse-based products, an excellent source of information is the book

[“The Java Developer’s Guide to Eclipse”, Second Edition, by Jim D’Anjou, Scott Fairbrother, Dan Kehn, John Kellerman and Pat McCarthy \(Addison-Wesley, ISBN 0-321-30502-7\).](#)

For those mostly interested in writing Eclipse plug-ins, another good source of information is the book

[“Eclipse – Building Commercial-Quality Plug-ins”, Second Edition, by Eric Clayberg and Dan Rubel \(Addison-Wesley, ISBN 0-321-42672-X\).](#)

For a quick guide to the Eclipse IDE, a very useful reference is the book

[“Eclipse IDE Pocket Guide”, by Ed Burnette \(O’Reilly Media, ISBN 0-596-10065-5\)](#)

Finally, a wealth of information is available in the [articles section](#) of the [Eclipse.org](#) web site as well as the [Eclipse Newsgroups](#). For information specific to Carbide.c++ one can visit the [Carbide.c++ discussion board](#) in [Forum Nokia](#) and the Symbian Tools newsgroup discussion on [symbian.tools](#).

[Back to Developer Library](#)

Want to be kept informed of new articles being made available on the Symbian Developer Network?

[Subscribe to the Symbian Community Newsletter.](#)

The Symbian Community Newsletter brings you, every month, the latest news and resources for Symbian OS.