

# Carbide.c++: Introductory White Paper

Version 1.0; March 20, 2006

Carbide

**NOKIA**

Copyright © 2006 Nokia Corporation. All rights reserved.

Nokia and Forum Nokia are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

#### **Disclaimer**

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

#### **License**

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

## Contents

<b>1</b>	<b>Introduction</b> .....	<b>5</b>
<b>2</b>	<b>Carbide Tools</b> .....	<b>6</b>
<b>3</b>	<b>Nokia and Eclipse</b> .....	<b>7</b>
<b>4</b>	<b>Carbide and Eclipse</b> .....	<b>8</b>
4.1	Introduction to the Eclipse IDE.....	8
4.2	Introduction to Carbide.c++ .....	9
<b>5</b>	<b>Developing with Carbide.c++</b> .....	<b>11</b>
5.1	Getting Started.....	11
5.2	Working with the Project .....	12
5.3	Notes on Using Carbide .....	14
5.4	Configuring Carbide.....	15
<b>6</b>	<b>Summary</b> .....	<b>18</b>
<b>7</b>	<b>Further Information</b> .....	<b>19</b>

## Change History

March 20, 2006	Version 1.0	Initial document release
----------------	-------------	--------------------------

# 1 Introduction

User demand for smart mobile applications, content, and services is growing. To meet these demands, developers need tools that allow them to create content and applications for mobile devices quickly and efficiently. Recognizing this demand, in late 2005, Nokia launched a new set of development environment tools called Carbide, a family of mobile software and content development tools that support several mobile technologies.

This document provides the following:

- A description of Nokia's overall strategy for the Carbide range of tools (Chapter 2).
- A review of Nokia's involvement with the Eclipse Foundation and an introduction to the Eclipse-based Carbide tools (Chapter 3).
- A look at the relationship between the Eclipse integrated development environment (IDE) and Carbide.c++ (Chapter 4).
- An introduction to using Carbide.c++ to create applications for the S60 platform (Chapter 5).

The document concludes with a list of where to find more information on Eclipse, Eclipse for C/C++ development, Carbide tools, and Eclipse plug-ins.

## 2 Carbide Tools

Mobile devices offer third-party developers, operators, and device manufacturers a range of ways to incorporate unique functionality, personalization, and branding into devices. Individual developers may be involved in a single element of this process, but many organizations are involved in all aspects of the mobile device experience. Each group of developers within this community has its own requirements for the tools its members use:

- Device creation — Here tools are used to look deep into the device and provide the developer with visibility into the operating system and the platform. Developers in this area demand a time-to-market advantage from their tools.
- Device integration — Working closely with the device creation teams, integrators customize the platform to fulfill a device's unique market proposition and develop drivers for its hardware features. This group focuses on performance tools.
- Application development — Developers working to add device or after-market features look for the easiest methods to develop and deploy customization content and applications on production phones. Productivity tools are the emphasis for these developers.
- In-market customization — Operators comprise an often-overlooked critical group of developers. They need a set of tools that allow them to customize devices simply and intuitively.

For all these developers, a set of tools based on common technology and providing a consistent set of functions with a standardized look and feel would offer many benefits. For this reason, Nokia has rationalized its tools offerings under the Carbide brand and has built many of these new tools using Eclipse.

To address the needs of developers, the Carbide offering has been organized into three product families:

- Tools for the development of Java™ applications — Carbide.j.
- Tools for the development of C++ applications for Symbian OS — Carbide.c++ and Carbide.vs.
- Tools for device customization and UI design tools, which have yet to be announced.

With the exception of Carbide.vs (which is designed to provide developers with a set of tools consistent with Microsoft Visual Studio .NET 2003), all the Carbide tools are based on the Eclipse IDE to provide a common look and feel.

### 3 Nokia and Eclipse

Eclipse is an open source project with wide industry support. The project focuses on providing an extensible development platform and application framework for building software. It is built from three components: the Eclipse platform, a set of Java development tools, and a plug-in environment that uses plug-in modules to change the tool's behavior.

Nokia has embraced Eclipse as the basis of several of its Carbide developer tools precisely because of its open and extensible nature. Eclipse makes it easy for Nokia to create tools that match the needs of developers working with Nokia's platforms and technologies. In addition, Eclipse is rapidly becoming a de facto industry standard for application development. Many developers, having used it for development on other platforms and technologies — notably Java technology — are already familiar with Eclipse. Eclipse as the basis of Nokia's developer tools, therefore, offers the familiarity of a widely used tool. This familiarity simplifies working with new technology.

To date, Nokia has introduced two Eclipse-based developer tools:

- Carbide.j is a tool designed specifically to help Java developers create mobile Java applications efficiently using Java™ Platform, Micro Edition (Java™ ME) for the Series 40 Platform, the S60 platform, and the Series 80 Platform.
- Carbide.c++ Express is a complete tool for Symbian OS C++ developers creating applications for the S60 platform, the Series 80 Platform, and UIQ.

A significant benefit of using Eclipse as the basis for these Carbide tools is that it allows Nokia to provide the first cost-free, fully integrated Symbian OS C++ IDE. Carbide.c++ Express is available for developers to download free of charge from [www.forum.nokia.com/carbide](http://www.forum.nokia.com/carbide). Carbide.c++ will also be available in two commercial versions:

- Carbide.c++ Developer Edition, a version with additional application-development tools, has been designed for developers building applications with more-stringent quality and performance requirements.
- Carbide.c++ Professional Edition is a version designed for Symbian OS licensees creating Symbian OS devices or for developers working with licensees to create fundamental software components for devices.

Carbide.c++ will also be available in a version for developers who create phone devices. The details of this product have not yet been announced.

While the initial release of these Carbide tools provides complete functionality to each target group of developers, the extensible nature of Eclipse means that Nokia is readily able to extend the capabilities of Carbide.j and Carbide.c++ to support other development technologies and more-advanced development tools. For example, with the release of Carbide.c++ Developer, Nokia adds on-device debugging and a rapid application development (RAD) feature.

In addition, Nokia expects to introduce additional Eclipse-based Carbide tools for customization and UI design.

## 4 Carbide and Eclipse

This section offers a brief look at the “standard” Eclipse Java IDE before explaining how Nokia has expanded the basic Eclipse framework to create Carbide.C++.

### 4.1 Introduction to the Eclipse IDE

Eclipse is the leading Java development environment. In this default mode, it is a powerful tool that incorporates all the components usually associated with an IDE. Figure 1 shows the Eclipse IDE. The display is broken into several elements:

- An *explorer window*, which allows developers to walk through the various components of their software.
- An *editor window*, where source code is edited.
- An *outline window*, which displays the components of the software being worked on (in the figure, variables, and methods derived from the code).
- An *output window*, which displays the output from the various tools in use (for example, compilers).

This display is very flexible; windows can be maximized to take over the entire display or hidden completely from view, and the windows can be completely rearranged. Layouts can be saved in configurations called *perspectives*, and a developer can switch perspectives by clicking the perspective button, located on the right-hand side of the menu bar. A preconfigured debugging perspective allows the developer to examine code through windows displaying stack traces, variable values, and other debugging information.

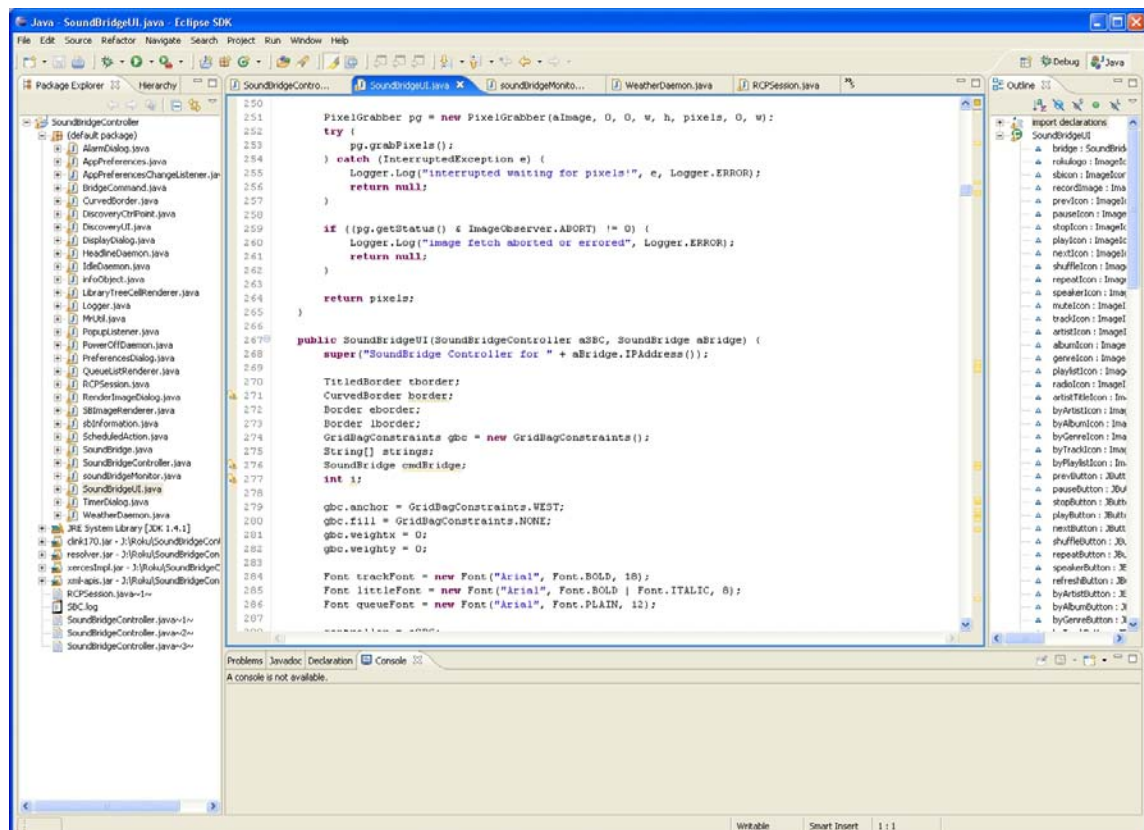


Figure 1: The standard Eclipse installation is a Java IDE.

Eclipse is an open source project, built to support plug-in modules. There are many examples of these plug-in modules, which can affect almost all aspects of Eclipse's behavior. Some plug-ins implement small behavioral changes (for example, the way tabs are turned into spaces), and some implement large behavioral changes (determining, for example, which editor is used or how programming files are parsed). Plug-ins also allow for implementation of new behavior via the Eclipse framework — for example, new menu functionality or new file-format support.

It is Eclipse's open, plug-in architecture that is used in the Carbide family to transform Eclipse into a highly effective set of development environments. Because the Carbide tools are created on a common base, the various language-development tools form a coherent family with a common look and feel. This means that developers who choose to work with multiple technologies are saved the trouble of learning multiple tool interfaces.

## 4.2 Introduction to Carbide.c++

Carbide.c++ takes advantage of the Eclipse IDE framework through plug-in modules. Nokia and Symbian Ltd. created new plug-ins to support development for Symbian OS, which replace the default set of plug-ins that created a Java IDE. These plug-ins change the standard IDE behavior as well as add new tools. Figure 2 shows Carbide.c++. Notice how the general look of Carbide matches the standard look of Eclipse as shown in Figure 1. The windows in the figure match those in Figure 1 — but with a C++ context. Many of the operational aspects of Carbide.c++ are the same as those of Eclipse.

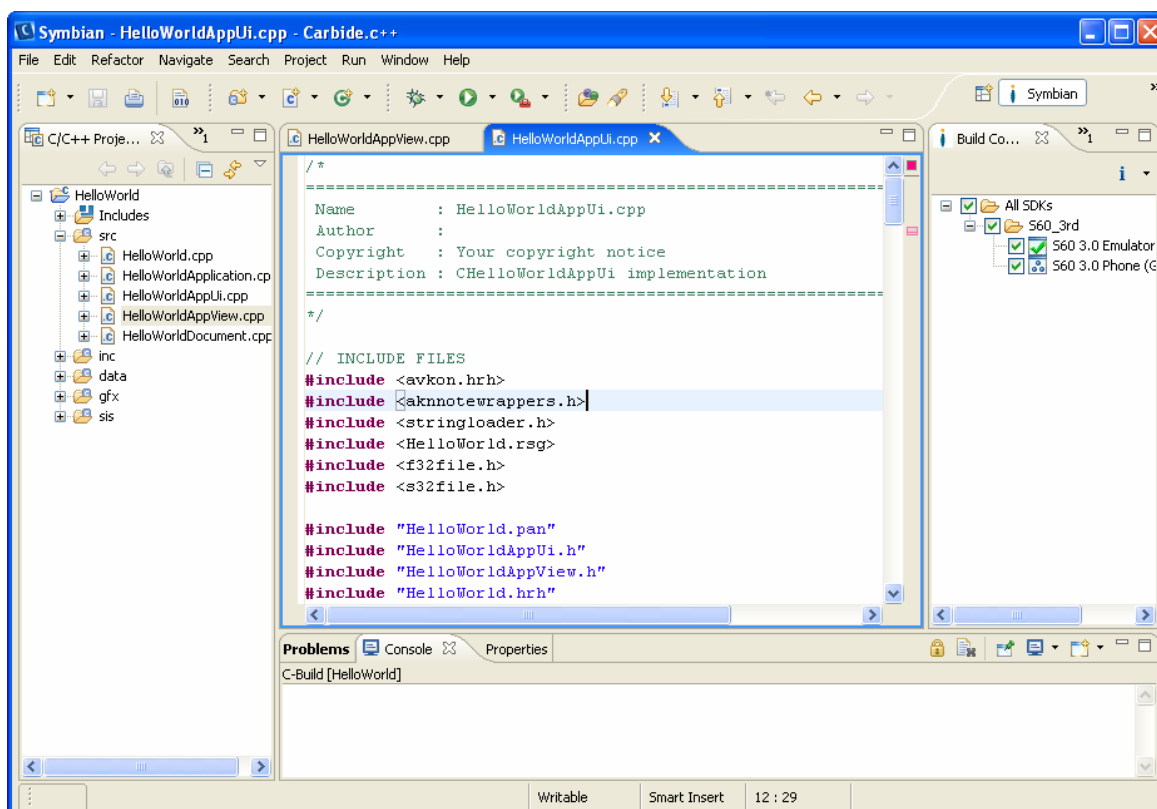


Figure 2: The Carbide IDE looks very much like Eclipse.

However, Carbide.c++ has also changed many aspects of Eclipse. The look and feel is the same, but the IDE is purely C++. From menu entries to editor configurations and SDK associations, Carbide.c++ has converted a Java IDE into a Symbian OS C++ IDE. Carbide.c++ adds a new "Symbian" perspective (see the upper right-hand corner in Figure 2) that includes support for build configurations and Symbian-specific files such as the Application Information File (AIF) and Multiple Bitmap (MBM) files.

To achieve this, many of the Eclipse plug-in modules have been replaced or removed. The default Eclipse IDE has 101 plug-in modules defined for it; Carbide.c++ has 119. The new plug-in structure is illustrated in Figure 3. There are fundamental plug-ins that match those in the default IDE. Symbian provides core Symbian OS functional support, mostly in the form of build sequences for the various versions of the operating system software. On top of Symbian support, Nokia provides a set of plug-ins. These provide build support for the various SDKs — for example, the S60 platform and UIQ — and provide for new debugger and compilation behavior. Finally, there is a set of plug-ins from Freescale Semiconductor, Inc. that implement CodeWarrior® technology.

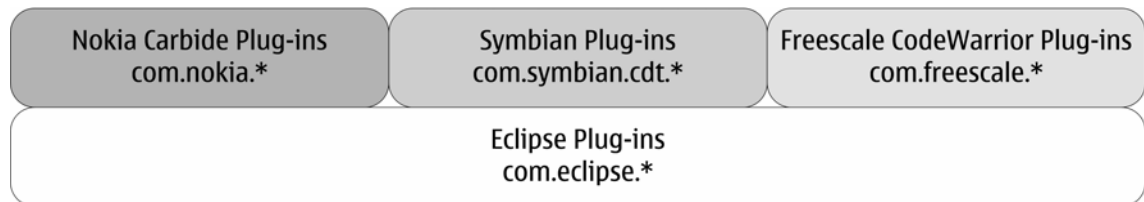


Figure 3: Carbide.c++ plug-ins come from several sources.

Using Eclipse for Carbide.c++ has several advantages. Nokia has supported the development of Eclipse, initially providing tools and SDKs for Java ME plug-ins that interfaced with Eclipse. The company has experience in using and programming the Eclipse plug-in interface. In addition, with the support of Symbian and the collaboration of Freescale for CodeWarrior behavior, the Eclipse IDE technology has been precisely targeted at Symbian OS C++ development.

## 5 Developing with Carbide.c++

As an introduction to the features of Carbide.c++, this section shows how an application can be built from the templates supplied with Carbide.c++. This section assumes that Carbide.c++ Express has been installed and is ready for use. Carbide.c++ Express can be downloaded for free from [www.forum.nokia.com/carbide](http://www.forum.nokia.com/carbide).

### 5.1 Getting Started

Carbide.c++ has a large number of wizards that help a developer start a new project. To begin, select **File > New** from the menu bar, and the menu choices, as shown in Figure 4, will be presented. Selecting **Other...** displays the complete selection of wizards.

After the appropriate initial project has been selected, Carbide.c++ walks the developer through the steps needed to create its starter files. For example, consider starting a S60 project. After specifying the name of the project and where the files will be stored, the developer is presented with a choice of templates, as shown in Figure 5. For an S60 application, the developer can choose from several types of user interface — from a very basic console-based application to one based on a GUI with S60 control components. The **S60 3.x GUI application** is a minimal GUI application in the Hello World-style with some code that shows how to use the S60 UI.

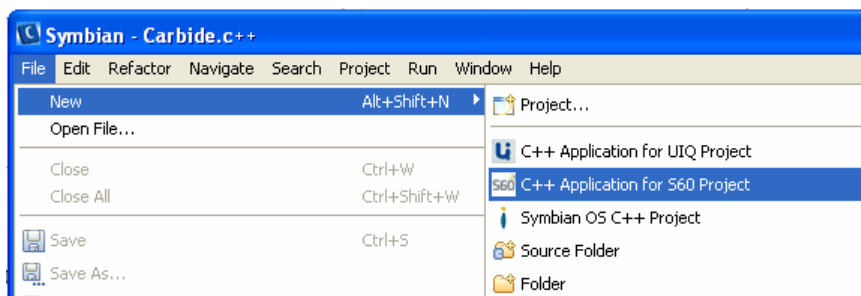


Figure 4: There are several choices for new projects in Carbide.c++.

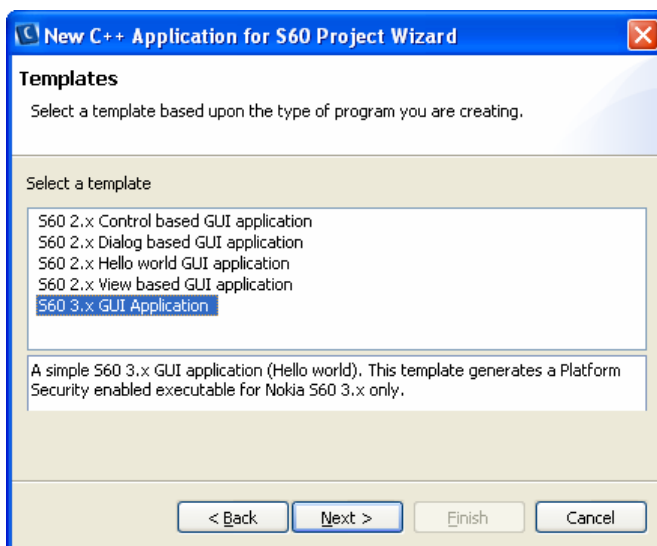


Figure 5: The developer may choose from several project templates in Carbide.c++.

Once the developer has chosen a template, a new dialog box is displayed that allows the developer to choose the appropriate SDKs. In the example illustrated in Figure 5, **S60 3.x GUI Application** is chosen (for S60 3<sup>rd</sup> Edition) from the dialog box, and two configurations — one for emulator builds and the other for device builds — are chosen in the dialog box that follows.

At this point, the developer can either click the **Finish** button to accept the remainder of the project defaults or click the sequence of **Next** buttons to walk through those defaults and fine-tune the project's configuration. When the **Finish** button is finally clicked, Carbide.c++ constructs the project by creating all the necessary files, with a bare minimum of code in them. Figure 6 shows the project in the Carbide.c++ Navigator window, with folders open to display some of the newly created files.

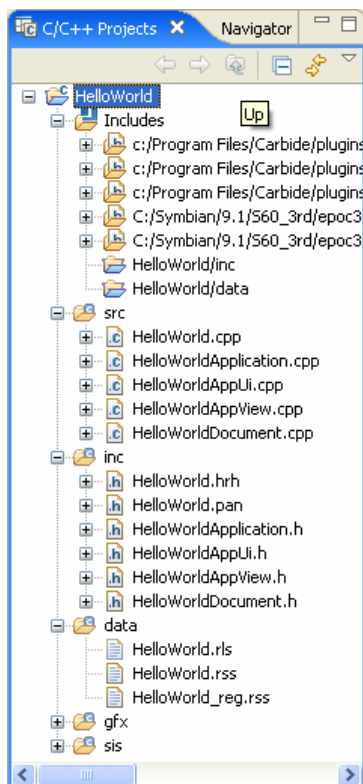


Figure 6: Carbide.c++ creates a complete project with template files.

## 5.2 Working with the Project

At this stage, there is a working application produced by the skeleton code files created by Carbide.c++. It is now the developer's job to fill in the code that implements an application. Among the many things a developer must do, the following basic set of duties must be performed:

- The source files, which are in the `/src` folder, must be edited in order to implement the project. All the included files — including `*.h` and `*.hrh` files — are in the `/inc` folder, and they, too, must be edited appropriately. The full-featured editor has standard text-editing functionality. This includes the concept of “code assist,” by which the editor provides the developer with choices about variable usage and API calling.

- Building a project is undertaken automatically. To build a project without running it, the developer can select **Project > Build All** from the toolbar. However, the most convenient way to build projects is to run or debug them. The build process is managed through the **C/C++ Build** options of the project's **Properties**, as shown in Figure 7. Here, every aspect of the build — from the build configuration, emulator, or device, through application-signing keys — is defined.

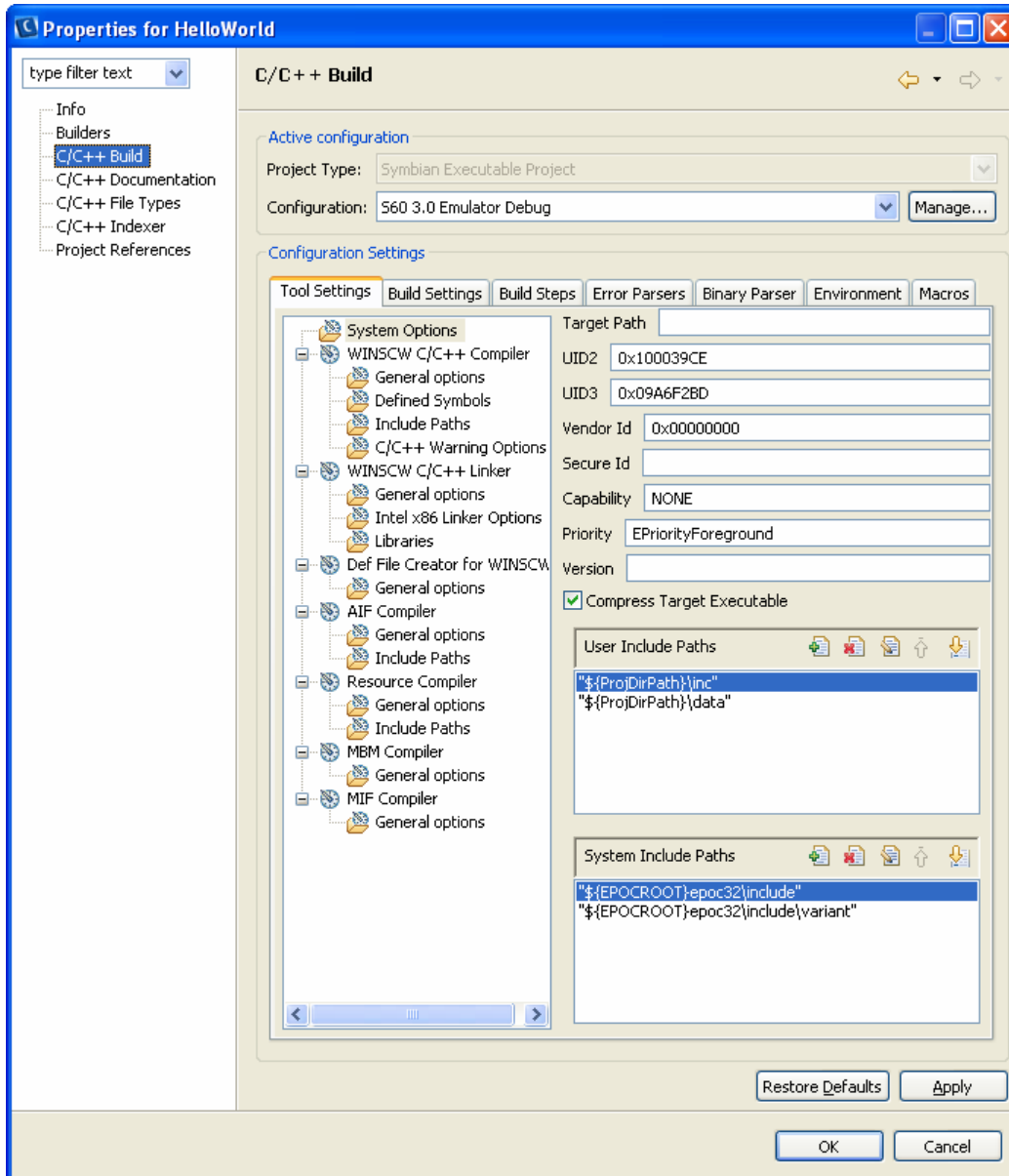


Figure 7: The **C/C++ Build** options control all aspects of the application build for multiple configurations.

- Running a project requires specification of a “launch configuration,” which is a collection of parameters — such as the name of the project to use, the location of the emulator, and the environment variables — that must be set. The launch configuration for the example used is shown in Figure 8.

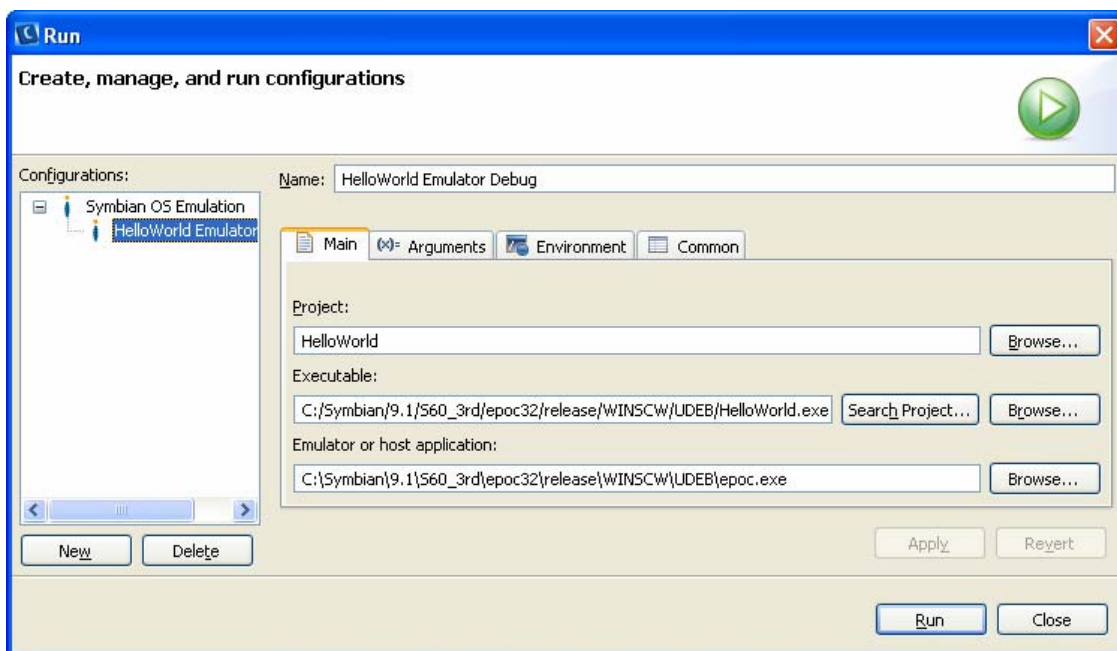


Figure 8: Running an application requires the creation of a launch configuration.

- Once a launch configuration has been specified, the application under development can be run. Running an application is started by clicking the green arrow on the toolbar.
- Debugging is initiated by clicking the “bug” icon on the toolbar. The full-featured debugger built into Carbide.c++ includes stepwise execution, breakpoints, and variable-value watching. In addition, Carbide.c++ Professional will include a debugging kernel that installs on a phone and allows for on-device debugging.

### 5.3 Notes on Using Carbide

There are a few important things to take note of when first using Carbide.c++.

- Signing Symbian OS applications:** Any application written for Symbian OS v9 and above must be “signed” by the developer before it can be installed on a Symbian OS phone. Carbide.c++ takes this into account in the Symbian Installation System (SIS) file-creation process. The keys used for signing an installation package are defined in the project’s **Properties**; if no keys are defined, Carbide.c++ automatically creates a set of self-signing keys.

- *Importing projects:* Existing projects can be imported into Carbide by selecting **File > Import**, which displays the dialog box shown in Figure 9. Complete projects can be imported using either the **Symbian Bld.inf** option or individual components imported with the **Symbian MMP File** option.

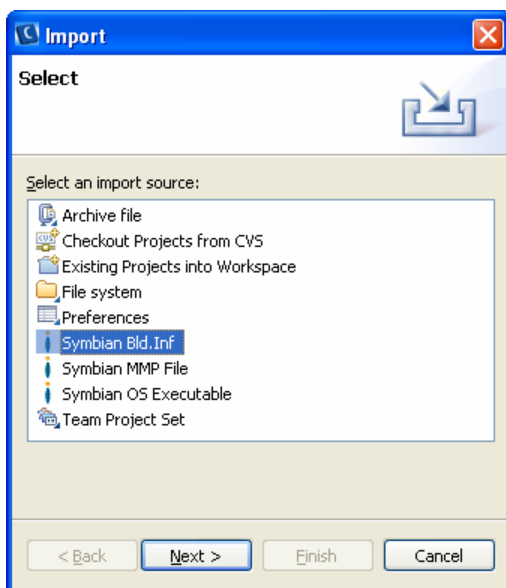


Figure 9: The **Import** dialog box allows for the importing of existing Symbian OS projects.

- *Plug-ins:* Most Eclipse plug-ins work with Carbide.c++. There are several plug-in repositories on the Internet. The Eclipse project's official repository is at [www.eclipse.org/downloads/index\\_topic.php](http://www.eclipse.org/downloads/index_topic.php). There are several third-party repositories, too, including Plugin Central at [www.eclipseplugincentral.com](http://www.eclipseplugincentral.com) and the EclipsePlugins site at [www.eclipse-plugins.info/eclipse/plugins.jsp](http://www.eclipse-plugins.info/eclipse/plugins.jsp). Installation of plug-ins is straightforward. The Eclipse Foundation ([www.eclipse.org](http://www.eclipse.org)) provides an SDK for writing plug-ins.
- *External tools:* Carbide.c++ inherits Eclipse's ability to interface with external tools. This allows extra functionality to be used for building, analyzing, and manipulating source files. To launch an external tool, a configuration, similar to that needed for applications, is required. External tools can also be added as part of the build process for a project. These external tools run in the specified order every time a project is built.

#### 5.4 Configuring Carbide

The initial Carbide.c++ configuration allows developers to start application development immediately, but there are several configuration options that allow them to adjust Carbide.c++ to suit their individual preferences. To make adjustments, display the IDE preferences window by selecting **Window > Preferences...** from the menu bar.

Carbide.c++ has a large set of adjustable parameters. The following are some of the most useful:

- Colors and fonts associated with syntax highlighting can be adjusted under the **General > Appearance > Colors and Fonts** selection. The Basic category is shown in Figure 10. Carbide.c++ offers many possibilities for syntax highlighting.

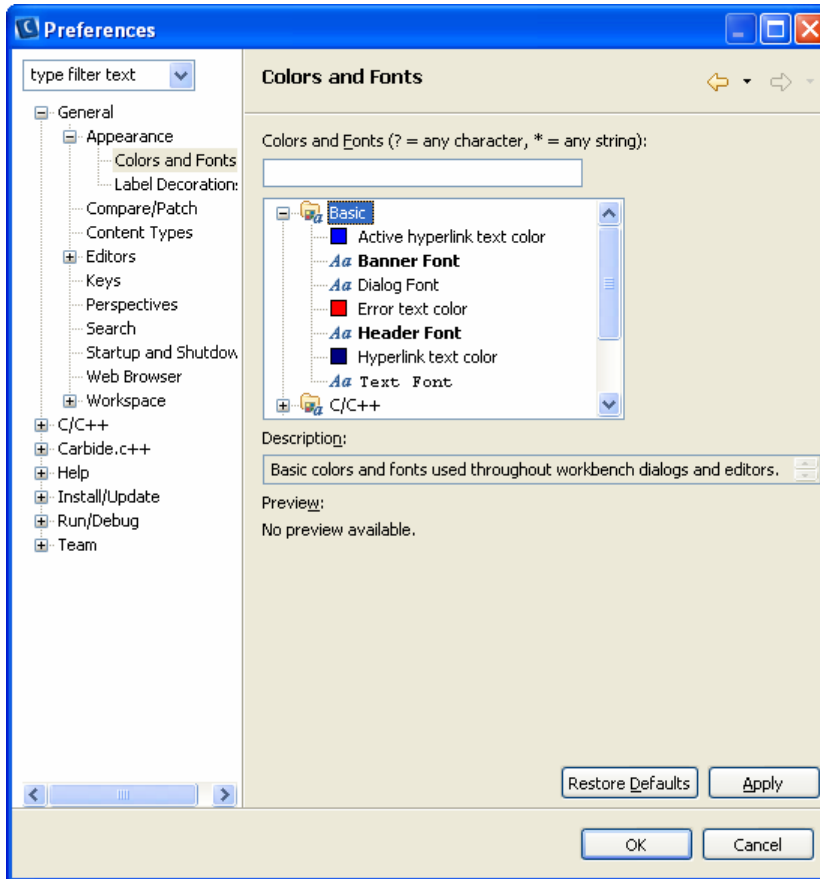


Figure 10: Basic **Colors and Fonts** preferences allow changes to be made to displayed text.

- “Code Assist” is a concept that Carbide.c++ uses to help developers sort through the many APIs that are available. Carbide.c++ presents the developer with a list of APIs or a set of variable definitions when certain patterns are typed into the editor. This assistance can be fine-tuned under the **C/C++ > Editor > Code Assist** section, shown in Figure 11. Often, users accustomed to other editing systems will want to adjust settings under the “Auto Activation” section. For example, making the delay a bit longer allows for faster typing when the code wanted is known.

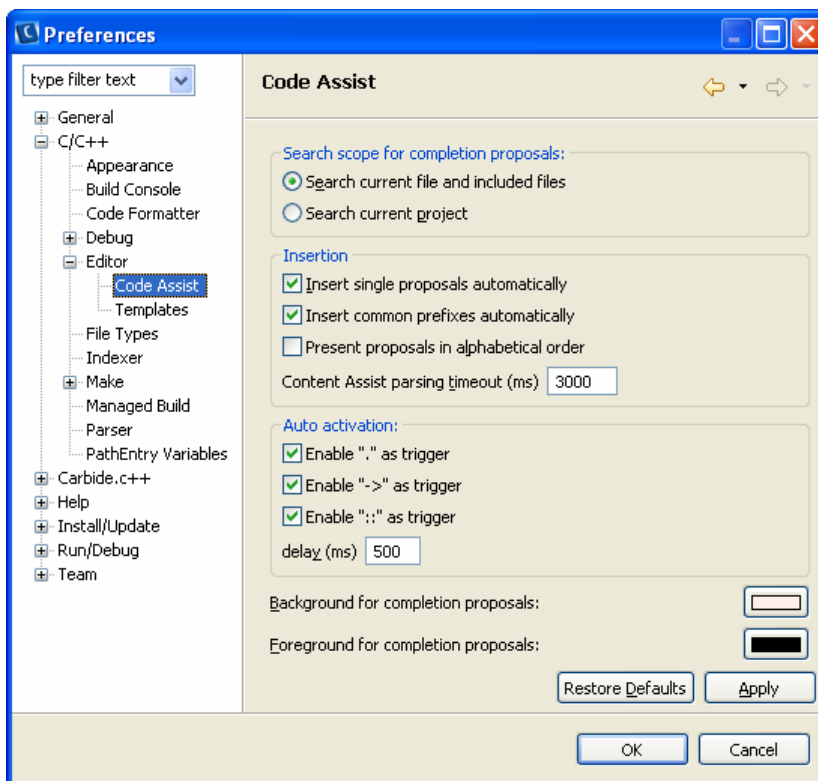


Figure 11: The **Code Assist** preferences allow the developer to tailor how Carbide.c++ assists in API and variable completion.

- Path names and variable value settings for building code can be adjusted under the **C/C++ > Managed Build** section.
- New SDKs can be added and old ones removed through the **Carbide.c++ > SDK Preferences** section. Installed SDKs can be set so that Carbide.c++ ignores them, and the option to automatically look for new SDKs can be controlled.
- Console message colors are configurable through the **Run/Debug > Console** setting.

## 6 Summary

This paper introduces Carbide, a family of tools for Java and C++ development on Nokia platforms. Carbide is based on the Eclipse IDE framework and provides Java and Symbian OS C++ developers with a single common interface that can be tailored for several different development targets. The paper also looks at Carbide.c++, a Symbian OS C++ IDE based on the Eclipse IDE framework. Through its Eclipse base, Carbide.c++ implements an extensible plug-in mechanism, which has been used to create a full-featured C++ development environment. This development environment allows developers to concentrate on code creation and testing by automating the entire build process for both the emulator and devices.

Nokia developers stand to benefit from these new tools. As the IDE develops, new features for Nokia platform-based development will be added to Carbide: Specifically, Symbian OS C++ developers will get on-device debugging and RAD in Carbide.c++ Developer and Professional. In addition, an extensive set of third-party plug-ins allows developers to extend the Carbide tools in new and unique ways.

## 7 Further Information

More information on Carbide.c++ and Eclipse can be found as follows:

- Eclipse Foundation — [www.eclipse.org](http://www.eclipse.org).
- The Eclipse CDT (C/C++ Development Tools) Project — [www.eclipse.org/cdt/](http://www.eclipse.org/cdt/).
- Forum Nokia's Carbide Development Tools home page — [www.forum.nokia.com/carbide](http://www.forum.nokia.com/carbide).
- Plug-ins:
  - The Eclipse project's repository — [www.eclipse.org/downloads/index\\_topic.php](http://www.eclipse.org/downloads/index_topic.php).
  - Plugin Central — [www.eclipseplugincentral.com](http://www.eclipseplugincentral.com).
  - Eclipse Plugins — [www.eclipse-plugins.info/eclipse/plugins.jsp](http://www.eclipse-plugins.info/eclipse/plugins.jsp).