Creating Projects Manually

Manual project creation is always an available option. It is more time-consuming, but it provides you more control over your project's contents and settings. Your understanding of the project's internals allows you to adjust settings of auto-created projects and to repair/reconfigure them in case of any trouble

Creating the Project

As mentioned at the start of Creating and Configuring Projects with Wind River Tornado, every C++test project is a CDT project; C++test just adds its own extensions/settings to every C/C++ project.

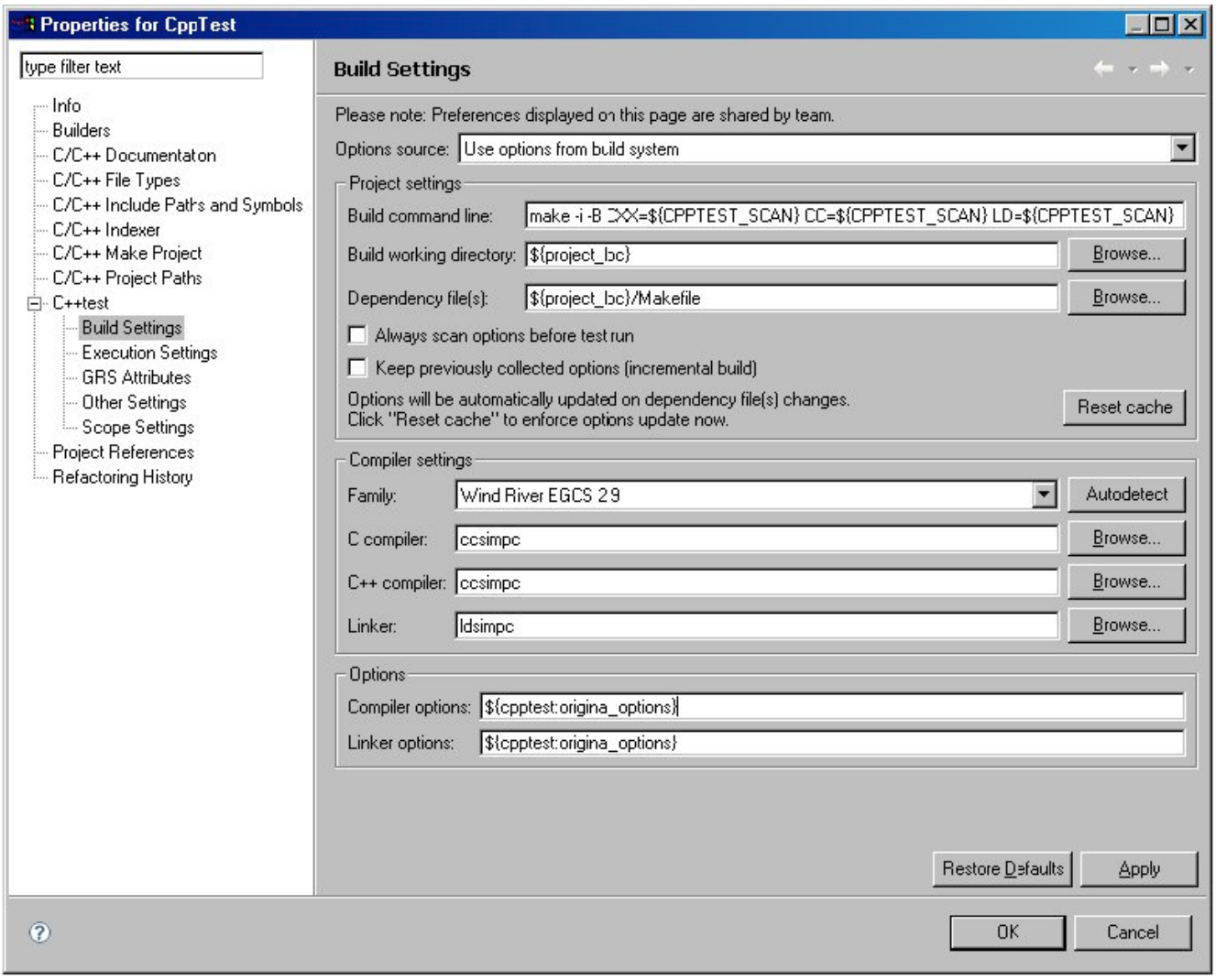
Configuring the Project

Next, you need to configure the appropriate build settings. This configuration is described in detail in Setting Project and File Options. Here, we will focus on those properties that are essential for configuring a Tornado-based project.

To review and modify settings:

1. Right-click the project tree node for the project whose settings you want to review and modify, then choose **Properties** from the shortcut menu. The Properties dialog will open.
2. Select **Parasoft> C++test> Build Settings** in the left pane.
3. Review the following settings and modify them if needed. The settings are described below.

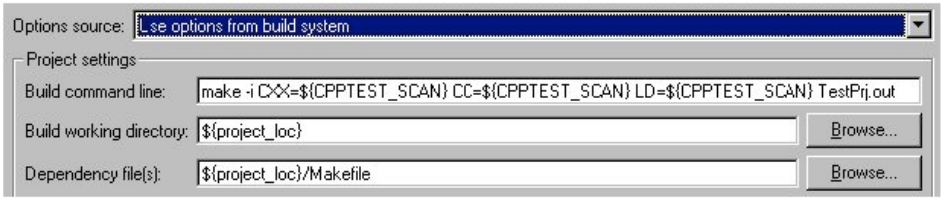
The first setting of concern is the Options Source setting. You can choose either **Use options from Wind River Tornado project** or **Use options from a build system**. The additional options available vary according to the selected option source.



Settings for "Use options from a build system"

When a project is first created, **Use options from a build system** is set by default as its options source. The following options are used to specify the project settings:

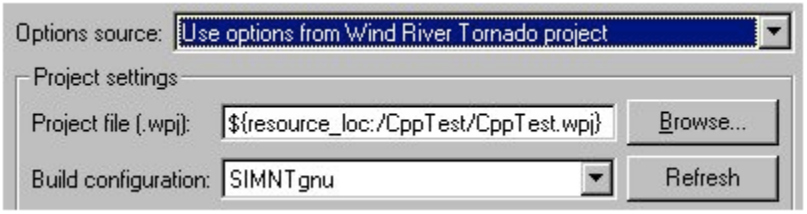
- **Build command line**: Enter a command line that will launch your build system; this system can be abstract, but by default, the make-based one is assumed and the appropriate command is preset. This command executes your make on the project's Makefile to scan the project's compilation and linking options. Thus, a special scanner (hidden behind the **${CPPTEST_SCAN}** macro) is substituted for C/C++ compilers and the linker; the **-i and -B make options** are very useful (see Accounting for Make Varieties for more information); you must also choose which one of the **Tornado project's rules** you want to launch as a make target (see Working with Tornado Projects' Rules).
- **Build working directory:** Specify the directory from which the build command is launched.
- **Dependency file(s):** Specify all files that should be checked for changes each time a test/build action is performed. If one of these files is found to be modified, the build command will be re-executed. You should enter all files that are sources of project's options (or influence them). Typically, this is just your Makefile.



Settings for "Use options from Wind River Tornado project"

If you'd rather extract options directly from your Tornado project file, set the options source to **Use options from Wind River Tornado project.** The following options are used to specify the project settings:

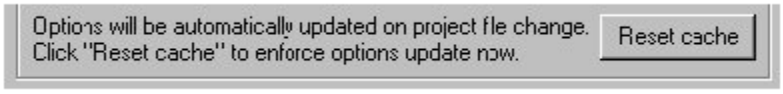
- **Project file (.wpj):** Specify a path to the Tornado project file.
- **Build configuration**: Choose from available build configurations read from the project file. To re-scan the configurations, click **Refresh.**



Settings for Both Options Source Types

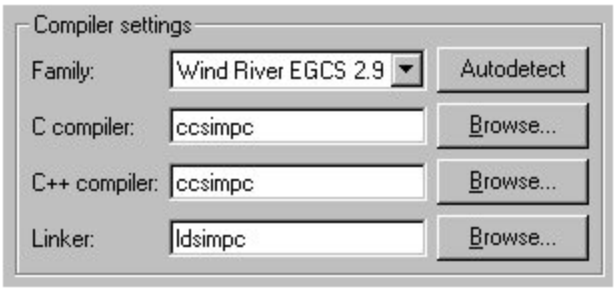
The remaining build settings are common to both types of option sources.

The **Reset cache** button lets you clear all scanned options and force rescanning on the next test action (see Accounting for Make Varieties for exceptions).



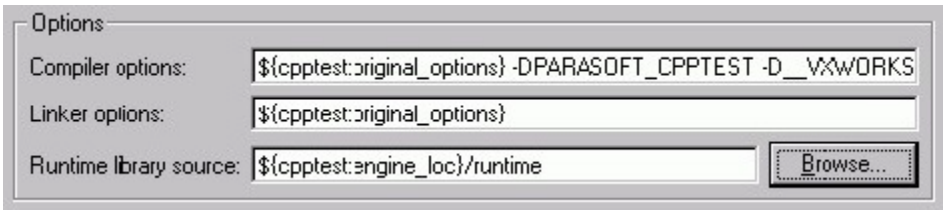
The **Compiler settings** area lets you specify the compiler set/tools used for compiling the project's sources and building the test executable (which we call the test relocatable, in the case of Tornado/VxWorks). Settings include:

- **Family:** Select the appropriate family from all the currently registered compiler families.
  - Families designed for Tornado are listed in Supported Tornado Versions and Compilers
  - You can duplicate/import compiler families using the Custom Compiler wizard, which is described in Adding a Custom Compiler Definition.
  - Click the **Autodetect** button to have C++test automatically detect the compiler family based on the compiler executables set and the compiler version regular expression that is stored in the compiler's configuration directory (due to the wide variety of compilers used with embedded solutions, this action may not always produce the expected result).
- **C compiler:** Specify the C compiler executable.
- **C++ compiler:** Specify the C++ compiler executable.
- **Linker:** Specify the linker executable.



The **Options** area lets you specify additional compilation and linking options; these are rarely used for regular host-based testing, but can be helpful for embedded testing. Settings include:

- **Compiler options:** The default value is `${cpptest:original_options} - DPARASOFT_CPPTEST`. You need to append this with necessary platform dependent options (see Setting Target/Platform Dependent Options) in order to perform testing.
- **Linker options:** The default value is `${cpptest:original_options}`. If you will be performing unit testing, you need to append this with the path to the C++test runtime library (see Understanding and Building the Runtime Library to learn how to build the runtime library).
- **Runtime library source:** The default value is `"${cpptest:engine_loc}/runtime.` This can be adjusted to provide alternate Runtime Library sources (as described in Understanding and Building the Runtime Library).



Accounting for Make Varieties

Every make tool is designed to interpret and execute Makefiles, but the features, options and behavior of different make versions may vary depending on the vendor, build, and even the direct application of the tool. When you use make, it's important to be aware of its capabilities and drawbacks to avoid surprises. Often, it's difficult to diagnose problems by monitoring the make's output—especially when you know it works with another version.

The following two make options are used extensively in C++test, but may be unavailable in particular versions of make:

- **-i** - instructs make to continue in spite of any errors returned from Makefile commands.
- **-B** - orders make to call targets unconditionally—resolving dependencies, but ignoring timestamp differences.

If your make doesn't support the `-i` option, then you must ensure that your build command will execute with no errors. Otherwise, you won't be able to scan options for all your project's files.

If the make doesn't support the `-B` option, then C++test will not rescan a previously-built project when one of the dependency files changes or when you click **Reset cache**—unless either you touch the appropriate source (or all sources), or clean the project. The make is run, but it reports up-to-date kind of messages and doesn't execute anything. In this case, it might be useful to write special make targets that touch/clean things on the fly as scanning proceeds.

It is important to note that **Tornado's GNU make-3.74 doesn't support the -B option!**

Working with Tornado Projects' Rules

To build a project, Tornado executes special default or user-defined rules specified in it.

The default rules cannot be modified. Their names and number depend on the Tornado project type. For the downloadable application module (recommended for testing with C++test), there are three default rules:

- **<project_name>.out:** Builds the test executable (relocatable).
- **objects:** Performs compilation only.
- **archive:** Performs compilation and archives the resulting objects.

Unlike the default rules, user-defined rules can be added, edited, or removed.

Rules exist independently from build configurations. Each build configuration has one rule selected to execute. Those rules are represented in Tornado-generated Makefiles.
You can build your project from the command line by executing `"make <target>"` from within the project's directory, where the target may come from a default or user-defined rule. (Remember that project Makefiles are regenerated when the Tornado environment executes the build command, so they may be outdated compared to the project ".wpj" file). The same applies to the C++test project build command. In most cases, you just need to append `"<project_name>.out"`