PARASOFT.
C++test™
A Parasoft AEP Technology™

C++test 7.0 Getting Started Guide

PARASOFT®

Introduction

Parasoft C++test is an integrated solution for automating a broad range of best practices proven to improve software development team productivity and software quality. C++test enables coding policy enforcement, static analysis, comprehensive code review, and unit and component testing to provide teams a practical way to ensure that their C and C++ code works as expected. C++test can be used both on the desktop under leading IDEs as well as in batch processes via command line interface for regression testing. C++test integrates with Parasoft's GRS reporting system, which provides interactive Web-based dashboards with drill-down capability, allowing teams to track project status and trends based on C++test results and other key process metrics.

C++test can help development teams:

- Apply a comprehensive set of best practices for identifying and addressing defects from the earliest phases of the development cycle—when fixing them requires minimal effort and rework.
- Automatically vet known coding issues so more time can be dedicated to tasks that require human intelligence.
- Efficiently construct, continuously execute, and maintain a comprehensive regression test suite that detects whether updates break existing functionality.
- Gain instant visibility into C and C++ code quality and readiness by accessing on-demand objective code assessments and tracking progress towards quality and schedule targets.
- Build an efficient, consistent, and controlled team workflow for applying best practices that reduce testing time, testing effort, and the number of defects that reach QA.
- Automate negative testing on a broad range of potential user paths to uncover problems that might otherwise surface only in "real-world" usage.

Choosing the C++test Version that Suits Your Development Environment

C++test is available in multiple versions to accommodate different development environments. The best version to use depends on your IDE:

If you are using this IDE...	Use this version of C++test...
Microsoft Visual Studio .NET 2003 or 2005	Visual Studio plugin
Eclipse CDT	Eclipse plugin
Wind River Workbench	Eclipse plugin
Anything else (including Visual C++ 6.0)	Eclipse standalone

C++test does not integrate with Visual Studio 2005 Express Edition.

C++test standalone includes software developed by the Eclipse Project (http://www.eclipse.org/).

Supported Environments

This section outlines the environments that are supported by C++test 7.0.

Sections include:

- Host (Development Machine) Compilers
- Build Utilities
- Embedded Platforms and Cross-Compilers

Host (Development Machine) Compilers

Platform	Compiler	Comments
Microsoft Windows NT/2000/XP	GNU and MingW gcc/g++ 2.95.x GNU gcc/g++ 3.2.x, 3.3.x, 3.4.x	For details on GCC support and a list of unsupported GCC compiler extensions, see the "GCC Support" C++test User's Guide topic.
	Microsoft Visual C++ 6.0 Compiler version (cl): C/C++ Optimizing Compiler Version 12.00.x Linker version (link): Incremental linker Version 6.00.x	Support for importing Visual Studio 6.0 projects is available. See the "Importing a Visual Studio 6.0 Project" C++test User's Guide topic.
	Microsoft Visual C++ .NET Compiler version (cl): C/C++ Optimizing Compiler Version 13.00.x Linker version (link): Incremental linker Version 7.00.x	N/A
	Microsoft Visual C++.NET 2003 Compiler version (cl): C/C++ Optimizing Compiler Version 13.10.x Linker version (link): Incremental linker Version 7.10.x	C++test's full functionality is available as a plugin for Visual Studio .NET 2003.
	Microsoft Visual C++ 2005 Compiler version (cl): C/C++ Optimizing Compiler Version 14.00.50727.42 for 80x86 Linker version (link): Incremental linker Version 8.00.50727.42	C++test's full functionality is available as a plugin for Visual Studio .NET 2005.
	Green Hills MULTI for Windows x86 Native v4.0.x	Support for reading MULTI 4.x projects is available. Coverage for unit testing is not supported.

Platform	Compiler	Comments
Linux kernel 2.4 or 2.6 or higher with glibc 2.2 or higher and an x86-compatible processor	GNU gcc/g++ 2.95.x, 3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x	For details on GCC support and a list of unsupported GCC compiler extensions, see the "GCC Support" C++test User's Guide topic.
Linux x86-64 kernel 2.6 or higher with glibc 2.3 or higher and an x86_64-compatible processor	GNU gcc/g++ 3.4.x, 4.0.x, 4.1.x	A set of 32 bit compatibility libraries is required (C++test is 32-bit, but can process 64-bit code). For details on GCC support and a list of unsupported GCC compiler extensions, see the "GCC Support" C++test User's Guide topic.
Solaris 7, 8, 9, 10 UltraSparc Processor	Forte Developer 6 Update 2 (6.2) Sun cc: Sun Workshop 6 Update 2 Version 5.3 Sun CC: Sun Workshop 6 Update 2 Version 5.3	This is known as Sun CC 5.3.
	Sun ONE Studio 8 Sun cc: Sun C 5.5 Sun CC: Sun C++ 5.5	This is known as Sun CC 5.5.
	Sun Studio 9 Sun cc: Sun C 5.6 Sun CC: Sun C++ 5.6	This is known as Sun CC 5.6.
	Sun Studio 10 Sun cc: Sun C 5.7 Sun CC: Sun C++ 5.7	This is known as Sun CC 5.7.
	Sun Studio 11 Sun cc: Sun C 5.8 Sun CC: Sun C++ 5.8	This is known as Sun CC 5.8.
	GNU gcc 2.95.x, 3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x	For details on GCC support and a list of unsupported GCC compiler extensions, see the "GCC Support" C++test User's Guide topic.
	Green Hills MULTI for SPARC Solaris Native v4.0.x	Support for reading MULTI 4.x projects is available. Coverage for unit testing is not supported.

Build Utilities

- GNU make
- Sun make
- Microsoft nmake
- JAM

- Any build scripts that can provide an option of overriding a compiler via an environment variable

Embedded Platforms and Cross-Compilers

The platforms and compilers discussed in this section apply specifically to embedded testing, which implies generation (and possibly validation) of tests on host, and execution of tests on a target board. If users perform only host-based testing of embedded code, platform/compiler support is determined by the main support table at the beginning of this document.

In most cases, using a cross-compiler requires preparing specific project configurations based on the built-in configurations for host-based compilers (GCC or Green Hills).

If not specified otherwise, full code coverage analysis is supported.

Platform	Compiler	Comments
Embedded Linux	GCC 2.95.x - 4.1.x	N/A
Wind River	GCC 3.4.x DIAB 5.4+	Wind River Tornado projects are not supported. Specific options for building the runtime library on VxWorks need to be configured.
Green Hills MULTI IDE 4.0.x / Integrity OS	Green Hills Optimizing compilers	Support for reading MULTI projects is available. Coverage analysis is not supported. Note that MULTI has built-in coverage analysis which can be used.

Installation and Licensing

This section describes the installation and licensing of C++test for Windows, Linux, and Solaris systems.

For details on how to perform silent ("headless") installation, see the “Using Silent Installation to Streamline Team-Wide C++test Installation and Setup” topic in the C++test User’s Guide (in the “Team-Wide Deployment Overview” section).

Standalone - Windows

Prerequisites

System Requirements

- Windows NT, 2000, XP.
- 1.5 GHz or higher x86 processor.
- 512 MB RAM (1 GB is recommended).
- A supported compiler or cross-compiler.
    - See “Supported Environments” on page 3 for a list of supported compilers.

Other Requirements

- For unit testing coverage, Parasoft Insure++ version 7.0.8 P05 or higher must be installed and licensed, and the PATH must be changed to include the directory with the Insure++ executables. If you do not already have Insure++, contact your Parasoft representative to obtain Insure++ with a special license for C++test coverage support.
- Additional disk space for C++test project data.
- The recommended Japanese language encoding is Shift_JIS (ja_JP.PCK locale on Solaris/Unix). Other encodings might cause font problems or prevent C++test from reading test results.

Installation

To install the standalone version of C++test on a Windows system:

1. In Windows Explorer, locate and double-click the self extracting archive.
2. Click **Yes** when a dialog asks whether you want to install C++test.
3. Click **Yes** after you have read and agreed with the license information.
4. Click **Next** after you have read the readme file.
5. Enter the desired installation directory in the Installation Location dialog box, then click **Next**. C++test will then start copying files. A dialog box with a progress indicator will open and indicate installation progress. When the installation is complete, a notification dialog box will open.
6. Click the **OK** button to close the notification dialog box.

C++test will be installed in the specified installation directory.

Startup

Before launching C++test

For C++test to autodetect compiler and makefile settings, the necessary executables (compiler/linker, makefile, etc.) must be correctly configured. "Correctly configured" means different things for different compilers, but it typically involves ensuring that the executable is on the PATH.

To launch the standalone, do one of the following:

- Launch the cpptest executable. Include the appropriate location (install dir) on the $PATH or launch it with the full path to the executable.
- Double-click the C++test desktop shortcut.

After Eclipse is launched, you should see a **C++test** menu added to the Eclipse menu bar. If you do not see this menu, choose **Window> Open Perspective> Other,** select **C++test**, then click **OK**.

If you suspect that C++test is not properly installed, see the "Troubleshooting and FAQs" C++test User's Guide topic for help resolving some common installation problems.

Standalone - Linux/Solaris

Prerequisites

System Requirements

- One of the following platforms:
  - Linux kernel 2.4 or 2.6 or higher with glibc 2.2 or higher and an x86-compatible processor.
  - Linux kernel 2.6 or higher with glibc 2.3 or higher and an x86_64-compatible processor (32-bit compatibility package is required).
  - Solaris 7, 8, 9, 10 and an UltraSPARC processor.
- 512 MB RAM (1 GB is recommended).
- A supported compiler or cross-compiler.
  - See "Supported Environments" on page 3 for a list of supported compilers.
  - If you are using a cross-compiler that is not listed as supported, see the "Configuring Testing with the Cross Compiler" C++test User's Guide topic for information on how you can use it with C++test.

Other Requirements

- For unit testing coverage, Parasoft Insure++ version 7.0.8 P05 or higher must be installed and licensed, and the PATH must be changed to include the directory with the Insure++ executables. If you do not already have Insure++, contact your Parasoft representative to obtain Insure++ with a special license for C++test coverage support.
- Additional disk space for C++test project data.
- The recommended Japanese language encoding is Shift_JIS (ja_JP.PCK locale on Solaris/Unix). Other encodings might cause font problems or prevent C++test from reading test results.

Warning - Critical workaround for installations where some users have restricted write privileges

Known issues with the location of Eclipse configuration/cached data (https://bugs.eclipse.org/bugs/show_bug.cgi?id=54919) could prevent the Eclipse-based C++test standalone from starting properly.

To prevent this problem:

1. Open the `<INSTAL_DIR>/configuration/config.ini` file for editing.
2. Add the following line (the actual directory name can be changed):
   osgi.configuration.area=@user.home/EclipseConfigData

As a result, Eclipse should keep all its config data in $(HOME)/EclipseConfigData directory (please be sure to have full access-rights to that location).

Installation

To install the standalone version of C++test on a Linux or Solaris system:

1. Copy the distribution file to the target location.
2. Execute the following command to make the .sh file executable:
   `chmod +x cpptest_eclipse_<arch>_<version>.sh`

3. Execute the following command to run the installation script:
   `./cpptest_eclipse_<arch>_<version>.sh`
4. When the script requests confirmation, confirm that C++test should be installed in the current directory.
5. Follow the additional instructions given by the installation script.

C++test will be installed in the current directory.

Startup

Before launching C++test

For C++test to autodetect compiler and makefile settings, the necessary executables (compiler/linker, makefile, etc.) must be correctly configured. "Correctly configured" means different things for different compilers, but it typically involves ensuring that the executable is on the PATH.

To launch the standalone:

- Launch the cpptest executable. Include the appropriate location (install dir) on the $PATH or launch it with the full path to the executable.

After Eclipse is launched, you should see a **C++test** menu added to the Eclipse menu bar. If you do not see this menu, choose **Window> Open Perspective> Other,** select **C++test**, then click **OK**.

If you suspect that C++test is not properly installed, see the "Troubleshooting and FAQs" C++test User's Guide topic for help resolving some common installation problems.

Eclipse Plugin - Windows

This section covers installation into Eclipse. See “Wind River Workbench Plugin - Windows” on page 15 for details on installing the Eclipse plugin into Wind River Workbench.

Prerequisites

System Requirements

- Windows NT, 2000, XP.
- 1.5 GHz or higher x86 processor.
- 512 MB RAM (1 GB is recommended).
- A supported compiler or cross-compiler.
    - See “Supported Environments” on page 13 for a list of supported compilers.

IDE Requirements

- Eclipse SDK 3.1+ or Eclipse Platform Runtime 3.1+ (32-bit).
    - Available at http://download.eclipse.org/eclipse/downloads/
- Eclipse CDT 3.1+
    - Available at http://www.eclipse.org/cdt/downloads.php
- Java runtime environment (JRE) supported by Eclipse.

Installation of C++test and Jtest into the same Eclipse environment is not recommended; use separate Eclipse installations instead.

Other Requirements

- For unit testing coverage, Parasoft Insure++ version 7.0.8 P05 or higher must be installed and licensed, and the PATH must be changed to include the directory with the Insure++ executables. If you do not already have Insure++, contact your Parasoft representative to obtain Insure++ with a special license for C++test coverage support.
- Additional disk space for C++test project data.
- The recommended Japanese language encoding is Shift_JIS (ja_JP.PCK locale on Solaris/Unix). Other encodings might cause font problems or prevent C++test from reading test results.

Installation

To install the C++test plugin for Eclipse on Windows:

1. In Windows Explorer, locate and double-click the self extracting archive.
2. Click **Yes** when a dialog asks whether you want to install C++test.
3. Click **Yes** after you have read and agreed with the license information.
4. Click **Next** after you have read the readme file.
5. Enter the desired destination directory for the C++test Extension files, then click **Next**.
6. Enter your Eclipse installation directory, then click **OK**.
    - Choose the directory that contains startup.jar.
7. Close Eclipse if it is open, then click **OK** to close the dialog reminding you to close this program. C++test will then start copying files and installing the necessary files into the workbench. A dialog

box with a progress indicator will open and indicate installation progress. When the installation is complete, a notification dialog box will open.

8. Click the **OK** button to close the notification dialog box.

Startup

Before launching C++test

For C++test to autodetect compiler and makefile settings, the necessary executables (compiler/linker, makefile, etc.) must be correctly configured. "Correctly configured" means different things for different compilers, but it typically involves ensuring that the executable is on the PATH.

To launch the plugin:

- Launch Eclipse as normal.

Eclipse will automatically find the C++test plugin.

After Eclipse is launched, you should see a **C++test** menu added to the Eclipse menu bar. If you do not see this menu, choose **Window> Open Perspective> Other,** select **C++test**, then click **OK**.

If you suspect that C++test is not properly installed, see the "Troubleshooting and FAQs" C++test User's Guide topic for help resolving some common installation problems.

Eclipse Plugin - Linux/Solaris

This section covers installation into Eclipse. See “Wind River Workbench Plugin - Linux or Solaris” on page 17 for details on installing the Eclipse plugin into Wind River Workbench.

Prerequisites

System Requirements

- One of the following platforms:
  - Linux kernel 2.4 or 2.6 or higher with glibc 2.2 or higher and an x86-compatible processor.
  - Linux kernel 2.6 or higher with glibc 2.3 or higher and an x86_64-compatible processor (32-bit compatibility package is required).
  - Solaris 7, 8, 9, 10 and an UltraSPARC processor.
- 512 MB RAM (1 GB is recommended).
- A supported compiler or cross-compiler.
  - See “Supported Environments” on page 13 for a list of supported compilers.

IDE Requirements

- Eclipse SDK 3.1+ or Eclipse Platform Runtime 3.1+ (32-bit).
  - Available at http://download.eclipse.org/eclipse/downloads/
- Eclipse CDT 3.1+
  - Available at http://www.eclipse.org/cdt/downloads.php
- Java runtime environment (JRE) supported by Eclipse.

Installation of C++test and Jtest into the same Eclipse environment is not recommended; use separate Eclipse installations instead.

Other Requirements

- All users must be able to write to the Eclipse configuration directory. If all users cannot write to the current Eclipse configuration directory, then the location of that directory must be changed. To change the directory location, open the <INSTAL_DIR>/configuration/config.ini file, then add a line of the format
  `osgi.configuration.area=@user.home/EclipseConfigData`
  (Be sure to enter the appropriate location.) This configures Eclipse to keep all its configuration data in the `$(HOME)/EclipseConfigData` directory. You must have full access-rights to that location)
- For unit testing coverage, Parasoft Insure++ version 7.0.8 P05 or higher must be installed and licensed, and the PATH must be changed to include the directory with the Insure++ executables. If you do not already have Insure++, contact your Parasoft representative to obtain Insure++ with a special license for C++test coverage support.
- Additional disk space for C++test project data.
- The recommended Japanese language encoding is Shift_JIS (ja_JP.PCK locale on Solaris/Unix). Other encodings might cause font problems or prevent C++test from reading test results.

Warning - Critical workaround for installations where some users have restricted write privileges

Known issues with the location of Eclipse configuration/cached data (https://bugs.eclipse.org/bugs/show_bug.cgi?id=54919) could prevent Eclipse from starting properly once C++test is installed.

To prevent this problem:

1. Open the `<INSTAL_DIR>/configuration/config.ini` file for editing.
2. Add the following line (the actual directory name can be changed):
   `osgi.configuration.area=@user.home/EclipseConfigData`

As a result, Eclipse should keep all its config data in $(HOME)/EclipseConfigData directory (please be sure to have full access-rights to that location).

Installation

To install the C++test plugin for Eclipse on UNIX:

1. Extract the contents of the C++test package into your preferred location.
2. Execute
   `extinstall <ECLIPSE_DIR>`
   where `<ECLIPSE_DIR>` is the root directory of your Eclipse installation.

If you later want to uninstall C++test:

1. Execute
   `extunintstall`
   C++test will then be "unregistered" from Eclipse.
2. (Optional) Remove C++test files from the hard drive.

Startup

Before launching C++test

To allow C++test to autodetect compiler and makefile settings, ensure that the necessary executables (compiler/linker, makefile, etc.) are correctly configured. "Correctly configured" means different things on different compilers, but it typically involves ensuring that the executable is on the PATH.

To launch the plugin:

- Launch Eclipse as normal.

Eclipse will automatically find the C++test plugin.

After Eclipse is launched, you should see a **C++test** menu added to the Eclipse menu bar. If you do not see this menu, choose **Window> Open Perspective> Other,** select `C++test`, then click **OK**.

If you suspect that C++test is not properly installed, see the "Troubleshooting and FAQs" C++test User's Guide topic for help resolving some common installation problems.

Visual Studio Plugin

System Requirements

- Windows XP or Windows 2003 server
- 512MB memory minimum, 1G recommended
- 1GHz or faster processor
- Visual Studio .NET 2003 or Visual Studio 2005
- Microsoft .NET Framework 1.1+

Installation of C++test and .TEST into the same Visual Studio environment. If you are using both products, have C++test create separate registry suffixes during installation (rather than register itself into the main Visual Studio installation). When the separate registry key mode is selected, the C++test plug-in will be activated only when launched from the C++test shortcut in the Start menu. It will not be available when Visual Studio is launched in any other way.

Installation

To install the C++test VS plugin:

1. In Windows Explorer, locate and double-click the C++test executable.
2. Answer the questions asked by the installer.

Wind River Workbench Plugin - Windows

Prerequisites

- Wind River Workbench 2.5+
- Eclipse CDT C/C++ Development Toolkit (In addition to the parts of CDT included with Workbench)
    - For Workbench 2.6, the recommended version of CDT is 3.1.2 (latest). It is available at http://www.eclipse.org/cdt/downloads.php.
    - For Workbench 2.5, the recommended version of CDT is 3.0.1. It is available at http://download.eclipse.org/tools/cdt/releases/eclipse3.1/dist/3.0.1.

Important

- You must install the full version CDT C/C++ Development Toolkit before installing C++test. Wind River Workbench includes parts of the CDT, but this is not sufficient for C++test. If you install C++test in Workbench without the CDT upgrade as describe above, C++test menus are not going to be visible.
- If you have upgraded your Workbench from 2.5 to 2.6, you will most likely have both workbench-2.5 and workbench-2.6 directories in your WindRiver installation directory. You may need to temporarily rename your workbench-2.5 directory before the install and reset it after C++test installation.
- Be sure to install C++test in a path without spaces in the directory names. C:\Parasoft is one such good option for a root installation. Do not install in the default location of C:\Program Files.

Installation

To install the C++test Eclipse plugin into Wind River Workbench on Windows:

1. Run the C++test plugin installation executable.
2. When prompted, enter the target directory where C++test should be installed. To prevent runtime problems, select a location whose path does not contain spaces.
3. When the prompted, enter the Wind River Workbench installation directory. Specify the directory that contains the `workbench-<version_number>` subdirectory; this is typically `C:/WindRiver`. C++test will verify the specified location by checking whether the following path exists: `<Wind River Installation Root>/workbench-<version_number>/wrwb/platform/eclipse/startup.jar`. If it's found, C++test will try to create a link file to Workbench's embedded Eclipse. The link file will be located in `<Wind River Installation Root>/workbench-<version_number>/wrwb/platform/eclipse/links/com.parasoft.xtest.cpptest.link`, and it will contain the path to the C++test installation directory.

If the Workbench installation is successfully identified, the install wizard will display a message to that effect. If you don't see the message that Wind River workbench installation was found, please contact Parasoft technical support.

After the plugin is successfully installed, you can start Wind River Workbench (with the C++test plugin installed) by launching Workbench in the usual manner.

To open C++test:

- Choose **Window> Open Perspective> Other**, then select the **C++test** perspective.

Uninstalling

To uninstall the C++test Workbench plugin:

- Use the Control Panel's Add or Remove Program functionality or run the Windows C++test installation executable and select the Remove option

Never uninstall the C++test plugin by removing the file `workbench-<version_number>/wrwb/platform/eclipse/links/com.parasoft.xtest.cpptest.link`. If you do this, you will still need to manually remove the C++test installation, and the Windows registry's Install Shield information will remain.

Wind River Workbench Plugin - Linux or Solaris

Prerequisites

- Wind River Workbench 2.5+
- Eclipse CDT C/C++ Development Toolkit (In addition to the parts of CDT included with Workbench)
    - For Workbench 2.6, the recommended version of CDT is 3.1.2 (latest). It is available at http://www.eclipse.org/cdt/downloads.php.
    - For Workbench 2.5, the recommended version of CDT is 3.0.1. It is available at http://download.eclipse.org/tools/cdt/releases/eclipse3.1/dist/3.0.1.

Important

- You must install the full version CDT C/C++ Development Toolkit before installing C++test. Wind River Workbench includes parts of the CDT, but this is not sufficient for C++test. If you install C++test in Workbench without the CDT upgrade as describe above, C++test menus are not going to be visible.
- If you have upgraded your Workbench from 2.5 to 2.6, you will most likely have both workbench-2.5 and workbench-2.6 directories in your WindRiver installation directory. You may need to temporarily rename your workbench-2.5 directory before the install and reset it after C++test installation.
- Be sure to install C++test in a path without spaces in the directory names. C:\Parasoft is one such good option for a root installation. Do not install in the default location of C:\Program Files.

Installation

To install the C++test Eclipse plugin into Wind River Workbench on Linux or Solaris:

1. cd to the directory where you want to install the C++test plugin. When you start the installation process, C++test will try to install itself into the current directory. To prevent runtime problems, select a location whose path does not contain spaces.
2. Launch the installation process from the shell console by executing the following command: `cpptest_eclipse_plugin_<platform>_<version>.sh`
3. When prompted, enter the Wind River Workbench installation directory. Specify the directory that contains the `workbench-<version_number>` subdirectory; in typical installations, this is . `/usr/local/WindRiver`. C++test will verify the specified location by checking whether the following path exists: `<Wind River Installation Root>/workbench-<version_number>/wrwb/platform/eclipse/startup.jar`. If it's found, C++test will try to create a link file to Workbench's embedded Eclipse. The link file will be located in `<Wind River Installation Root>/workbench-version_number>/wrwb/platform/eclipse/links/com.parasoft.xtest.cpptest.link`, and it will contain the path to the C++test installation directory.

After the plugin is successfully installed, you can start Wind River Workbench (with the C++test plugin installed) by launching Workbench in the usual manner.

Uninstalling

To uninstall the C++test plugin, do one of the following:

- Call the `extunistall` executable located in the main C++test installation directory.
- Manually remove the link file (`<Wind River Installation Root>/workbench-verison_number>/wrwb/platform/eclipse/links/com.parasoft.xtest.cpptest.link`) that the installation added to the Wind River Workbench directory.

In both cases, you will need to manually remove C++test files.

Licensing

Using a Machine-Locked License

To install a machine- locked license:

1. Choose **C++test> Preferences** to open the Preferences dialog.
2. Select the **License** category in the left pane.
3. Contact your Parasoft representative to receive your license. You will need to provide the Machine ID listed in the Local License area.
    - In the United States, call 1-888-305-0041. In other locations, use the contact information available at "Contacting Parasoft" on page 28.
4. Enter your license expiration date and password in the Local License section of the License preferences page.
5. Click **Apply**. The License preferences page will display the features that you are licensed to use.
6. Click **OK** to set and save your license.

Using Parasoft LicenseServer

Setting the license if all users cannot write to the C++test installation directory

The user who has write access to the C++test installation directory should configure the license on behalf of all team members. If the license is set by a user who does not have write access to the C++test installation directory, the license information will be stored at the workspace level, and will need to be re-entered for each new workspace.

To install a license when the Parasoft LicenseServer (available separately) manages C++test licensing across your team or organization:

1. Choose **C++test> Preferences** to open the Preferences dialog.
2. Select the **License** category in the left pane.
3. Select the **Use LicenseServer** option. The LicenseServer section of the License preferences page will become active.
4. If the appropriate LicenseServer is not already set, select it from the **Autodetected servers** list and click **Set**. Or, manually enter your organization's LicenseServer host (either a name or an IP address) in the **Host name** field, then enter your organization's LicenseServer port in the **Port number** field.
5. Indicate the license type that you want this C++test installation to use by selecting the appropriate option in the **Edition** box. Available options are:
    - **Professional Edition:** Covers coding standards static analysis, unit testing, and embedded support.
    - **Architect Edition:** Covers coding standards static analysis, RuleWizard (custom rule creation), unit testing, and embedded support.
    - **Server Edition:** Covers coding standards static analysis, RuleWizard (custom rule creation), unit testing, BugDetective, the command-line interface, and embedded support.
    - **Professional Edition IT:** Covers coding standards static analysis and unit testing.

- **Architect Edition IT:** Covers coding standards static analysis, RuleWizard (custom rule creation), and unit testing.
- **Server Edition IT:** Covers coding standards static analysis, RuleWizard (custom rule creation), unit testing, BugDetective, and the command-line interface.
- **Custom Edition:** Covers custom licensing needs. If you are using a custom license, select this option, then click the **Choose** button and specify which of your available license features you want to apply to this installation.

6. Click **OK** to set and save your LicenseServer settings.

If your organization needs additional licenses or updated licenses, the manager or architect should contact Parasoft to obtain these licenses, then add these licenses to LicenseServer as described in the LicenseServer documentation.

Setting Up

Creating Custom Test Configurations

Every C++test test-- whether it is performed in the GUI or from the command line interface-- is based on a Test Configuration which defines a test scenario and sets all related test parameters for static analysis, test generation, and test execution. To change how a test is performed, you modify the settings for the Test Configuration you plan to use.

C++test provides built-in Test Configurations that are based on a variety of popular test scenarios. However, because development projects and team priorities differ, most C++test users prefer to create custom Test Configurations. For instance, you can create a custom Test Configuration that checks just the rules that your team has decided to follow, another one that generates test cases using your preferred generation settings, and another one that executes all available automatically-generated and user-defined test cases.

The default Test Configurations, which are in the Built-in category, cannot be modified. The recommended way to create a custom Test Configuration is to copy a Built-in Test Configuration to the User-defined category, then modify the copied Test Configuration to suit your preferences and environment. Alternatively, you could create a new Test Configuration "from scratch", then modify it as needed.

The Default Configuration should be set to the custom Test Configuration that you plan to use most frequently. By setting your preferred Test Configuration as the Default Configuration, you can easily run it from the **C++test** menu, the **Test Using** tool bar button, or from the command line interface.

Unless your team has already configured a Team Test Configuration for all team members to use, we recommend that you create a custom Test Configuration and set it as the Default Test Configuration.

To create a custom Test Configuration:

1. Open the Test Configurations panel by choosing **C++test> Test Configurations**.
2. Review the available Test Configurations to determine which (if any) you want to base your custom Test Configuration on.
3. Do one of the following:
    - If you want to base a custom Test Configuration on a built-in Test Configuration, right-click that Test Configuration, then choose **Duplicate**.
    - If you want to create a custom Test Configuration from scratch, click **New**.
4. Select the new Test Configuration, which will be added to the **User-defined** category.
5. Modify the settings as needed.
    - Scope tab settings determine what code is tested; it allows you to restrict tests by author, by timestamp, and so on.
    - Static tab settings determine how static analysis is performed and what rules it checks.
    - Generation tab settings determine how test cases are generated.
    - Execution tab settings determine how test cases are executed.
    - Common tab settings cover various actions that can affect multiple types of analysis.
    - Goals tab settings allow the team to specify goals for error reporting and error correction.
6. (Optional) Set the Test Configuration as the Default Test Configuration by right-clicking it, then choosing **Set as Default** from the shortcut menu. The configuration will then be set as the Default Configuration; the default icon will be added to that configuration in the Test Configurations tree.
7. Click **Apply**, then **Close**.

Creating Projects

Standalone or Eclipse Plugin

There are several ways to build a project. Use the following guide to select the option best suited to your needs:

- If you want to keep the source files and test files in separate directories, link to the source files.
- If you want to create multiple projects with the same set of source files (for instance, so you can maintain separate configurations for a host compiler and a target compiler), link to source files.
- If your source files are in CVS and you want the test files stored in the same location as the source files, use CVS.
- If your source files are NOT in CVS and you want the test files (automatically-generated settings files, test files, stub definitions, etc.) stored in the same directory as the project source, use the source directory as the project location.

In this getting started guide, we will cover how to link to the source files. For details on the other methods, see the "Creating a Project" topic in the C++test User's Guide

Linking to the Source Files

If you prefer to store your test files in a separate location than your source files-- or if you want to create multiple projects that use the same source files-- create a project as follows:

1. Create an empty project as follows:
    a. Choose **File> New> Standard Make C Project** or **File> New> Standard Make C++ Project**.
    b. Enter a name for the project in the **Project name** field.
    c. Click **Finish**.
2. Link your source files to that project as follows:
    a. Choose **File> New> Folder**.
    b. Select the name of the project that you created in Step 1.
    c. Click the **Advanced** button.
    d. Enable the **Link to folder in file system** option.
    e. Enter or browse to the location of your source files.
    f. Click **Finish**.

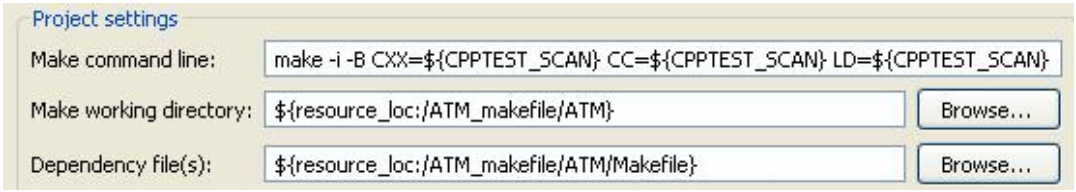
The linked files will appear within the project folder in the C/C++ Projects view and Navigator view, but they will continue to "live" within their original location. C++test will not add any new files to the linked source location; any files that it generates will be saved in the project directory.

Warning - Using Appropriate Build Settings

When configuring a project with a linked source folder, you need to modify any project's build settings that refer to `${project_loc}`; these settings must be changed to use `resource_loc` instead. If you do not change these default settings, the project's source files will be skipped during testing.

To change the project's build settings:

1. Right-click the C/C++ Projects tree node for the project, then choose **Properties** from the shortcut menu. The Properties dialog will open.
2. Expand the **C++test** category in the left pane.
3. Select the **Build Settings** category.
4. Modify the **Make working directory** setting to `${resource_loc:/<project name>/<linked source dir>/<actual make working dir path from there>}`.
5. Modify the **Dependency file(s)** setting to `${resource_loc:/<project name>/<linked source dir>/<actual make working dir path from there>/Makefile}`.



Visual Studio Plugin

Before you can test code with C++test, it must be available in a Microsoft Visual Studio project.

Verifying Build Settings

The appropriate build settings must be configured in order for C++test to properly test your code. Additionally, GRS and Advanced Settings (instrumentation options) can be configured as needed.

Before the initial test, we strongly recommend that you review settings for each project, then modify the settings as needed to suit your environment.

To review and modify settings:

1. In the project tree (in Eclipse, the C/C++ Projects view; in Visual Studio, the Solution Explorer), right-click the node for the project or file whose settings you want to review and modify, then choose **Properties** (Eclipse) or **C++test Properties** (VS) from the shortcut menu. The Properties dialog will open.
2. Select the category that represents the settings you want to review and/or change. Categories and available settings are described below.
    - Build settings must be reviewed and then modified as needed. Reviewing other settings is optional. All available settings are described in the "Setting Project and File Options" C++test User's Guide topic.
3. Modify the options in the right pane.
4. Click **Apply**, then **OK**.

Running a Test

Prerequisites

C++test Standalone or Eclipse Plugin

- You must be in the C/C++ or C++test perspective in order to run tests.
- Before you can test code with C++test, it must be added to an Eclipse C/C++ project. For instructions on creating a new project, see the "Setting Up" C++test User's Guide section.

Visual Studio Plugin

- Before you can test code with C++test it must be added to a Visual Studio project.

Common

- Before you perform the initial test, we strongly recommend that you review and modify project options. For details on how to do this, see the "Setting Project and File Options" C++test User's Guide topic.
- We recommend that you prepare a customized Test Configuration before you perform tests. For details on how to do this, see the "Creating Custom Test Configurations" C++test User's Guide topic.

With an appropriate license, tests can also be run from the command-line; see the "Testing in Command Line Interface" C++test User's Guide topic for details.

Testing from the C++test GUI

C++test can perform a variety of tasks, from static analysis, to regression test generation/execution, to exception finding, to checking functional unit test case. To start using C++test to achieve your goals, you run a test based on a default or custom test scenario, which defines the precise nature and scope of C++test's analysis. These test scenarios are called "Test Configurations," and they define test scope, static analysis, test case generation, and test case execution settings. .

The general procedure for testing from the GUI is as follows:

1. In the project tree (in Eclipse, the C/C++ Projects view; in Visual Studio, the Solution Explorer), select the resources(s) that you want to test. You can use **Ctrl + click** or **Shift + click** to select multiple resources.
2. Start the test in one of the following ways:
    - Choose the appropriate Test Configuration from the **Test Using** section of the **Test Using** button's pull-down menu.
    - Choose the appropriate Test Configuration from the **C++test> Test Using** menu in the menu bar.
    - Choose the appropriate Test Configuration from the **C++test> Test History** menu in the menu bar. Note that this menu contains only the most recently-run Test Configurations.
    - Right-click the selection, then choose the appropriate Test Configuration from the **C++test> Test Using** shortcut menu.
    - Right-click the selection, then choose the appropriate Test Configuration from the **C++test> Test History** shortcut menu.

C++test will then run the test scenario defined by the selected Test Configuration.

Unit Testing Procedure

For unit testing, we recommend that you run multiple Test Configurations in the following order:

- Generate Unit Tests
- Generate Stubs
- Build Test Executable
- Run Unit Tests

Reviewing Results

Test progress and results summaries will be reported in the Testing panel that C++test opens when it starts the test. Detailed results will be reported in the C++test view in the C++test perspective. Drill down to see details about the test findings. Coverage results are reported in the Coverage view.

In the C++test view, tasks are organized into the following categories:

- **Fix Static Analysis Violations:** This category contains static analysis violations.
- **Fix Unit Test Problems:** This category contains unit testing problems-- including functional test failures, unexpected exceptions, and timeouts.
- **Review Unit Test Outcomes:** This category contains unverified outcomes for test cases that were created during automated test case generation. Unverified outcomes are reported when C++test executes automatically-generated or user-defined test cases with postconditions that have not yet been converted to assertions. The outcome might be the expected behavior, or it might indicate a problem. Further review and verification is required. If you determine that the outcome reflects the expected behavior, you verify it. If not, you specify the correct outcome.

Customizing Static Analysis

Static analysis can be customized to:

- Change the rules checked during coding standard analysis.
- Modify coding standard static analysis settings, such as number of static analysis tasks reported per rule.
- Prevent the reporting of additional occurrences of a specific static analysis task.
- Create custom rules for checking project/team/organization-specific requirements, enforcing standard API usage, and preventing the recurrence of application-specific defects after a single instance has been found.
- Customize the built-in rules.
- Change rule categories, IDs, headers, and/or severities.
- Create new rule categories.

For details, see the "Customizing Coding Standards Static Analysis: Overview" C++test User's Guide topic.

Customizing and Extending the Test Suite

The test suite can be extended and customized to:

- Add new test cases to check specific unit-level functionality requirements or to improve coverage.
- Modify automatically-generated test cases to check specific unit-level functionality requirements or to improve coverage.
- Modify test generation or execution settings.
- Remove test cases and disable outcome checks or test cases that are not currently of concern to you.
- Prevent the testing of certain classes or methods.
- Convert automatically-generated tests into a "functional snapshot" for regression testing (to identify changes/problems introduce by code modifications).
- Access data source values during testing.
- Use standard I/O data in test cases.
- Define custom stubs for testing code in isolation and/or testing when certain functions are not available.
- Execute legacy CppUnit test cases using C++test .

For details, see the “Extending and Modifying the Test Suite: Overview” C++test User’s Guide topic.

Additional Resources

User Guides

C++test Standalone or Eclipse Plugin Plugin

The C++test documentation library includes the following items:

- **The C++test User's Guide:** Explains how to use the C++test functionality that is built upon Eclipse (if you have the standalone version of C++test) or that is added to Eclipse or Workbench (if you have the C++test plugin). To access this guide through the Eclipse/Workbench help system, choose **Help> Help Contents**, then open the **C++test User's Guide** book. The PDF is available in the `manuals` directory within the C++test installation directory.
- **The RuleWizard User's Guide**: Explains how to use RuleWizard, Parasoft's rule creation technology, to create custom rules that can check specific project and organizational requirements, or prevent application-specific errors from recurring. To access this guide, choose **C++test> Launch RuleWizard** to open RuleWizard, then choose **Help> Documentation** in the RuleWizard GUI. Note that RuleWizard requires a special license.
- **The C/C++ Coding Standards Rules Guide:** Describes all of the coding standard rules included with C++test. To access this guide through the Eclipse help system, choose **Help> Help Contents**, then open the **C++test Coding Standard Rules** book.

Additional user guides in the Eclipse help system (for example, the Workbench User Guide, C/C++ Development User Guide, etc.) describe native Eclipse and C/C++ Development Toolkit functionality and strategies.

Wind River Workbench Plugin Usage Details

For details on how to use C++test from within Wind River Workbench, see the "Wind River Plugin" section of the C++test User's Guide.

Visual Studio Plugin

The C++test documentation library includes the following items:

- **The C++test User's Guide:** Explains how to use the C++test functionality that is built into Visual Studio. To access this guide through the Visual Studio help system, choose **C++test> Help**, then open **C++test Documentation> User's Guide**. The PDF is available in the `manuals` directory within the C++test installation directory.
- **The RuleWizard User's Guide**: Explains how to use RuleWizard, Parasoft's rule creation technology, to create custom rules that can check specific project and organizational requirements, or prevent application-specific errors from recurring. To access this guide, choose C++test> Launch RuleWizard to open RuleWizard, then choose **Help> Documentation** in the RuleWizard GUI. Note that RuleWizard requires a special license.
- **The C/C++ Coding Standards Rules Guide:** Describes all of the coding standard rules included with C++test. To access this guide through the Visual Studio help system, choose **C++test> Help**, then open the **C++test Documentation> Rules Documentation**.

Forums

Parasoft's C++test Forum is an active, online meeting place where you can converse with and learn from fellow C++test users. Review existing threads, post your questions, and participate in the latest discussions at http://forums.parasoft.com.

Contacting Parasoft

C++test experts are available online to answer your questions. This live support allows you to chat in real-time with the C++test team and perform desktop sharing if needed. To receive live online support, go to http://www.parasoft.com/jsp/pr/live_experts.jsp. This live tech support feature currently supports only the Microsoft Windows operating system.

Contacting Parasoft Via Phone, E-Mail, or Fax

USA Headquarters

Tel: (888) 305-0041 or (626) 256-3680

Email cpptest-support@parasoft.com

France

Tel: (33 1) 64 89 26 00

Email: quality@parasoft-fr.com

Germany

Tel: +49 89 4613323-0

Email: quality@parasoft-de.com

UK

Tel: +44 (0)1923 858005

Email: quality@parasoft-uk.com

Asia

Tel: +886 2 6636-8090

Email: info-psa@parasoft.com

Other Locations

See http://www.parasoft.com/jsp/pr/contacts.jsp?itemId=268.