



User's Guide

Version 1.2

ParaSoft Corporation
2031 S. Myrtle Ave.
Monrovia, CA 91016
Phone: (888) 305-0041
Fax: (626) 305-9048
E-mail: info@parasoft.com
URL: www.parasoft.com

PARASOFT END USER LICENSE AGREEMENT

REDISTRIBUTION NOT PERMITTED

This Agreement has 3 parts. Part I applies if you have not purchased a license to the accompanying software (the "SOFTWARE"). Part II applies if you have purchased a license to the SOFTWARE. Part III applies to all license grants. If you initially acquired a copy of the SOFTWARE without purchasing a license and you wish to purchase a license, contact ParaSoft Corporation ("PARASOFT"):

(626) 305-0041

(888) 305-0041 (USA only)

(626) 305-9048 (Fax)

info@parasoft.com

<http://www.parasoft.com>

PART I -- TERMS APPLICABLE WHEN LICENSE FEES NOT (YET) PAID GRANT.

DISCLAIMER OF WARRANTY.

Free of charge SOFTWARE is provided on an "AS IS" basis, without warranty of any kind, including without limitation the warranties of merchantability, fitness for a particular purpose and non-infringement. The entire risk as to the quality and performance of the SOFTWARE is borne by you. Should the SOFTWARE prove defective, you and not PARASOFT assume the entire cost of any service and repair. This disclaimer of warranty constitutes an essential part of the agreement. SOME JURISDICTIONS DO NOT ALLOW EXCLUSIONS OF AN IMPLIED WARRANTY, SO THIS DISCLAIMER MAY NOT APPLY TO YOU AND YOU MAY HAVE OTHER LEGAL RIGHTS THAT VARY BY JURISDICTION.

PART II -- TERMS APPLICABLE WHEN LICENSE FEES PAID

GRANT OF LICENSE.

PARASOFT hereby grants you, and you accept, a limited license to use the enclosed electronic media, user manuals, and any related materials (collectively called the SOFTWARE in this AGREEMENT). You may install the SOFTWARE in only one location on a single disk or in one location on the temporary or permanent replacement of this disk. If you wish to install the SOFTWARE in multiple locations, you must either license an additional copy of the SOFTWARE from PARASOFT or request a multi-user license from PARASOFT. You may not transfer or sub-license, either temporarily or permanently, your right to use the SOFTWARE under this AGREEMENT without the prior written consent of PARASOFT.

LIMITED WARRANTY.

PARASOFT warrants for a period of thirty (30) days from the date of purchase, that under normal use, the material of the electronic media will not prove defective. If, during the thirty (30) day period, the software media shall prove defective, you may return them to PARASOFT for a replacement without charge.

THIS IS A LIMITED WARRANTY AND IT IS THE ONLY WARRANTY MADE BY PARASOFT. PARASOFT MAKES NO OTHER EXPRESS WARRANTY AND NO WARRANTY OF NONINFRINGEMENT OF THIRD PARTIES' RIGHTS. THE DURATION OF IMPLIED WARRANTIES, INCLUDING WITHOUT LIMITATION, WARRANTIES OF MERCHANTABILITY AND OF FITNESS FOR A PARTICULAR PURPOSE, IS LIMITED TO THE ABOVE LIMITED WARRANTY PERIOD; SOME JURISDICTIONS DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO LIMITATIONS MAY NOT APPLY TO YOU. NO PARASOFT DEALER, AGENT, OR EMPLOYEE IS AUTHORIZED TO MAKE ANY MODIFICATIONS, EXTENSIONS, OR ADDITIONS TO THIS WARRANTY.

If any modifications are made to the SOFTWARE by you during the warranty period; if the media is subjected to accident, abuse, or improper use; or if you violate the terms of this Agreement, then this warranty shall immediately be terminated. This warranty shall not apply if the SOFTWARE is used on or in conjunction with hardware or software other than the unmodified version of hardware and software with which the SOFTWARE was designed to be used as described in the Documentation. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY HAVE OTHER LEGAL RIGHTS THAT VARY BY JURISDICTION.

YOUR ORIGINAL ELECTRONIC MEDIA/ARCHIVAL COPIES.

The electronic media enclosed contain an original PARASOFT label. Use the original electronic media to make "back-up" or "archival" copies for the purpose of running the SOFTWARE program. You should not use the original electronic media in your terminal except to create the archival copy. After recording the archival copies, place the original electronic media in a safe place. Other than these archival copies, you agree that no other copies of the SOFTWARE will be made.

TERM.

This AGREEMENT is effective from the day you install the SOFTWARE and continues until you return the original SOFTWARE to PARASOFT, in which case you must also certify in writing that you have destroyed any archival copies you may have recorded on any memory system or magnetic, electronic, or optical media and likewise any copies of the written materials.

CUSTOMER REGISTRATION.

PARASOFT may from time to time revise or update the SOFTWARE. These revisions will be made generally available at PARASOFT's discretion. Revisions or

notification of revisions can only be provided to you if you have registered with a PARASOFT representative or on the ParaSoft Web site. PARASOFT's customer services are available only to registered users.

PART III -- TERMS APPLICABLE TO ALL LICENSE GRANTS

SCOPE OF GRANT.

DERIVED PRODUCTS.

Products developed from the use of the SOFTWARE remain your property. No royalty fees or runtime licenses are required on said products.

PARASOFT'S RIGHTS.

You acknowledge that the SOFTWARE is the sole and exclusive property of PARASOFT. By accepting this agreement you do not become the owner of the SOFTWARE, but you do have the right to use the SOFTWARE in accordance with this AGREEMENT. You agree to use your best efforts and all reasonable steps to protect the SOFTWARE from use, reproduction, or distribution, except as authorized by this AGREEMENT. You agree not to disassemble, de-compile or otherwise reverse engineer the SOFTWARE.

SUITABILITY.

PARASOFT has worked hard to make this a quality product, however PARASOFT makes no warranties as to the suitability, accuracy, or operational characteristics of this SOFTWARE. The SOFTWARE is sold on an "as-is" basis.

EXCLUSIONS.

PARASOFT shall have no obligation to support SOFTWARE that is not the then current release.

TERMINATION OF AGREEMENT.

If any of the terms and conditions of this AGREEMENT are broken, this AGREEMENT will terminate automatically. Upon termination, you must return the software to PARASOFT or destroy all copies of the SOFTWARE and Documentation. At that time you must also certify, in writing, that you have not retained any copies of the SOFTWARE.

LIMITATION OF LIABILITY.

You agree that PARASOFT's liability for any damages to you or to any other party shall not exceed the license fee paid for the SOFTWARE.

PARASOFT WILL NOT BE RESPONSIBLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OF THE SOFTWARE ARISING OUT OF ANY BREACH OF THE WARRANTY, EVEN IF PARASOFT HAS BEEN ADVISED OF SUCH DAMAGES. THIS PRODUCT IS SOLD "AS-IS".

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

ENTIRE AGREEMENT.

This Agreement represents the complete agreement concerning this license and may be amended only by a writing executed by both parties. THE ACCEPTANCE OF ANY PURCHASE ORDER PLACED BY YOU IS EXPRESSLY MADE CONDITIONAL ON YOUR ASSENT TO THE TERMS SET FORTH HEREIN, AND NOT THOSE IN YOUR PURCHASE ORDER. If any provision of this Agreement is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This Agreement shall be governed by California law (except for conflict of law provisions).

All brand and product names are trademarks or registered trademarks of their respective holders.

Copyright 1993-2001

ParaSoft Corporation

2031 South Myrtle Avenue

Monrovia, CA 91016

Printed in the U.S.A, April 27, 2001

Table of Contents

Introduction

Welcome to C++Test	1
Installation and Licensing	2

Testing With C++Test

Quick Start Guide

Quick Start Guide	7
-------------------------	---

Essential Tasks

Opening the File(s) You Want to Test	10
Building a Test Executable	13
Initializing Objects	14
Entering Test Cases	15
Running a Test	21
Testing a Method	24
Single-Stepping Through a Method	25
Testing a Class	26
Testing a File, Set of Files, or Developer Studio Project	27
Playing a Single Test Case	30
Test Results and Output	31
Validating Test Case Outcomes	41
Editing Source Code	42
Viewing Source Code	43
Refreshing Your Symbol Tree to Reflect Code Modifications	45
Performing Regression Testing	46

Optional Tasks

Working With Stubs	47
Excluding Specific Classes or Methods From Your Test	51
Enforcing Static Coding Standards With CodeWizard	53

Detecting Runtime Errors With Insure++	54
Monitoring Test Coverage	56
Monitoring Test Progress	59
Customizing General Test Settings	60
Customizing Project Test Settings	62
Customizing Automatic Test Settings	64
Working With C++Test Projects	67
Sharing Tests	69
Printing Results	70

C++Test GUI Help

GUI Overview	71
The Menu Bar	73
The Tool Bar	76
The File List	78
The Symbol Tree	79
The Source Code Tab	81
The Test Progress Tab	83
The Results Tab	85
The Test Case Editor Tab	88
The Stub Tables Tab	91
The Suppressions Tab	92
The Progress Tab	93
The Output Tab	94
The Messages Tab	95
The Settings Panels	96

Getting Help

If Your Build Fails	97
Contacting ParaSoft	99
C++Test Tutorials	101

Index

Index	103
-------------	-----

Welcome to C++Test

Unit testing is widely recognized as an effective way to improve application quality, but until now, C/C++ developers have not had a feasible way to perform unit testing. If you wanted to perform unit testing, you had to perform all steps-- from building harnesses, to building stubs, to building and running test cases-- by hand. This process is difficult, time-consuming, and expensive.

That's why ParaSoft developed C++Test, an automatic unit testing tool for C and C++. C++Test automates every part of unit testing that can possibly be automated. Specifically, it automatically...

- Builds a harness for each file that it tests.
- Creates any necessary stubs, and allows you to customize these stubs' return values or enter your own stubs.
- Performs all steps involved in white-box testing.
- Generates test cases that can be used as a base set of black-box test cases.
- Runs black-box test cases.
- Generates outcomes for black-box inputs.
- Performs regression testing.
- Tracks test coverage.

In addition, C++Test lets you run your classes through CodeWizard, for automatic error-prevention, and through Insure++, for automatic runtime error-detection.

Installation and Licensing



Installation


To install C++Test, simply run the setup executable that you downloaded from the ParaSoft Web site or that is on your CD, and follow the installation program's onscreen directions. The installation program will automatically install C++Test on your computer.

Important: Close Developer Studio before you install C++Test

Developer Studio Integration

C++Test will automatically install itself into Microsoft Developer Studio 6.0. After installation is complete, three C++Test buttons will be added to the Developer Studio tool bar:

Button	Name	Action
	Test Project	Automatically loads the current Developer Studio project into C++Test, builds all necessary test executables, and performs white-box testing on the project's files.
	Test File	Automatically loads the current Developer Studio file into C++Test, builds a test executable, and performs white-box testing on the file.

	C++Test	Opens the C++Test GUI.
-----------------------------------------------------------------------------------	----------------	------------------------

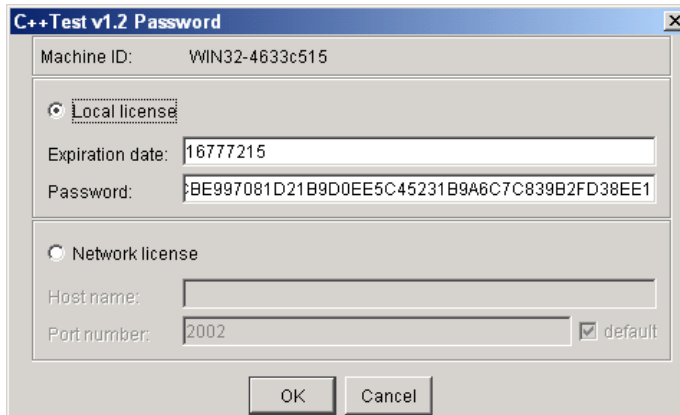
Note: You must have a project or file open in a Developer Studio workspace in order to use the **Test Project** and **Test File** commands.

Licensing

To install a machine-locked C++Test license on your machine:

1. Launch C++Test by either:
 - Clicking the **C++Test** tool bar button in Developer Studio.
 - Choosing **Start> Programs> C++Test> C++Test**.

A dialog box asking if you would like to install a password will open. Click **Yes**. The Password window will open.



The dialog box titled "C++Test v1.2 Password" contains the following fields and options:

- Machine ID: WIN32-4633c515
- ☒ Local license
 - Expiration date: 16777215
 - Password: BE997081D21B9D0EE5C45231B9A6C7C839B2FD38EE1
- ☐ Network license
 - Host name: [empty field]
 - Port number: 2002 ☒ default

Buttons: OK, Cancel

2. Call **1-888-305-0041** to get your license.
3. In the top portion of the Password window, enter your expiration date and password.
4. Click **OK** to set and save your license.

To install a network license and have ParaSoft's LicenseServer manage license access across your local area network:

1. Launch C++Test by either:
 - Clicking the **C++Test** tool bar button in Developer Studio.
 - Choosing **Start> Program Files> C++Test**.

A dialog box asking if you would like to install a password will open. Click **Yes**. The Password window will open.

The screenshot shows the 'C++Test v1.2 Password' dialog box. It has a title bar with the text 'C++Test v1.2 Password' and a close button. The dialog is divided into two main sections. The top section is for 'Local license' and the bottom section is for 'Network license'. In the 'Local license' section, there are two input fields: 'Expiration date:' and 'Password:'. In the 'Network license' section, there are two input fields: 'Host name:' (which contains the text 'hound.parasoft.com') and 'Port number:' (which contains the text '2071'). There is also a checkbox labeled 'default' next to the 'Port number' field. At the bottom of the dialog, there are two buttons: 'OK' and 'Cancel'.

2. In the Password window, select the **Network License** option.
3. Enter your LicenseServer Host name in the **Host Name** field.
4. Enter your LicenseServer port number in the **Port Number** field (the default port is 2002).
5. Click **OK** to set and save your LicenseServer information.
6. Call **1-888-305-0041** to get your license.

7. Add your license to LicenseServer as described in the LicenseServer documentation.

Quick Start Guide

To test your classes and functions with C++Test, perform the following steps:

1. Open the C or C++ file(s) or DevStudio project that you want to test:
 - a. Choose either of the following
 - **File> Open File(s)** (or click **Open File**).
 - **File> Open DevStudio Project**.
 - b. In the file chooser that opens, select the file(s) or project that you want to test.
 - c. Click **OK**.
2. Build a test executable and perform automatic white-box testing by clicking **Test All**.
3. View results in the Results tab. To see more details about an error found, select the red error node. Details about the error will be displayed in the Messages tab at the bottom of the GUI.
4. Validate outcomes of automatic test cases and correct any incorrect outcomes. To correct incorrect outcomes:
 - a. Select the test case in the Test Case Editor tab.
 - b. Click **Edit Test Case**.
 - c. Correct the appropriate values in the Test Case Editor window that opens. To edit a node's value, double-click it, then type or select the desired value.
 - d. Click **OK** to close the Test Case Editor window and modify the test case.
5. Add additional test cases by opening the Test Case Editor tab and performing the following steps for each test case that you want to add:
 - a. Select the Test Case Editor tree node associated with the method that you want your test case to test.

- b. Click **Add Test Case**.
 - c. Add test case values in the Test Case Editor window that opens. To add a value to a node, double-click it, then type or select the desired value.
 - d. Click **OK** to close the Test Case Editor window and add the new test case.
6. Run the new test cases, as well as the test cases that were automatically generated and executed during the previous run:
 - a. In the Symbol tree (middle left GUI pane), select the file, class, or method that you want to test.
 - b. Open the Test Progress tab.
 - c. Click **Play**.
7. View test results in the Results tab. To see more details about an error found, select the red error node. Details about the error will be displayed in the Messages tab at the bottom of the GUI.
8. In the Test Case Editor, validate the outcome of any test cases whose outcomes were automatically generated; if any of these test case produced an incorrect outcome, enter the correct reference value.
9. Edit your source code to correct problems found.
10. Ensure that your Symbol tree reflects source code modifications by clicking **Read Symbols** or **Build Test**.
11. Perform regression testing:
 - a. Open the Test Progress tab.
 - b. In the Symbol tree (middle left GUI pane), select the file, class, or method that you want to test.
 - c. Click **Play**.

You may also want to:

- Configure stubs or stub tables
- Exclude certain classes or methods from your test
- Enforce static coding standards with CodeWizard

- Detect runtime errors with Insure++
- Monitor test coverage
- View your source code
- Customize automatic test settings
- Save your files and parameters as a C++Test project

Opening the File(s) You Want to Test

There are several ways to open the file(s) or Microsoft Developer Studio (DevStudio) project that you want to test.

Previously Untested File(s)/DevStudio Projects

If you have not yet tested the file(s), there are two ways to open the file:

1. Simply open the file(s).
2. Open and test the file(s) in a single step.

Simply Open the File(s)

If you want to perform such tasks as viewing your source, adding test cases, or customizing automatic test parameters, you should open your file(s) by choosing **File> Open File(s)** or **File> Open DevStudio Project**, or by clicking **Open File**. This will open a file chooser from which you can choose the file(s) or DevStudio Project that you want to test.

Open and Test the File(s) in a Single Step

If you want to perform white-box testing on the file(s) or DevStudio Project immediately after you open it, you can save time by doing one of the following:

- Clicking the Developer Studio **Test File** button (to test a selected file in your Developer Studio workspace).
- Clicking the Developer Studio **Test Project** button (to test the entire project in your Developer Studio workspace).
- Choosing **Tests> Test File(s)** (to select and test files from the C++Test GUI).

- Choosing **Tests> Test DevStudio Project** (to select and test a DevStudio project from the C++Test GUI).

Any of these commands will prompt C++Test to:

- Open (or let you select) the file or Developer Studio project that you want to test.
- Build any necessary test executables.
- Automatically generate and execute white-box test cases.

Previously Tested File(s)/DevStudio Projects

To open a previously tested file or DevStudio Project, use any of the methods described in the above section, or choose the file/DevStudio Project from **File> Recent Files** or **File> Recent DevStudio Projects**.

If, during your previous tests of this class, you added any test cases or altered any of the class' or methods' settings, these modifications will be restored when you open the file(s)/DevStudio Project.

C++Test Projects

To open a C++Test Project, choose **Projects> Open** and select your project from the file chooser that opens. C++Test will then restore the project's files and parameters.

For more information on C++Test Projects, see "Working With C++Test Projects" on page 67.

Quick Reference Table

FORMAT	NOT YET TESTED	PREVIOUSLY TESTED
File(s)	<ul style="list-style-type: none"> • Choose File> Open File • Click Open File • Choose Tests> Test File(s) • Click the Test File Developer Studio button 	<ul style="list-style-type: none"> • Choose File> Recent Files • Choose File> Open File • Click Open File • Choose Tests> Test File(s) • Click the Test File Developer Studio button
DevStudio Project	<ul style="list-style-type: none"> • Choose File> Open DevStudio Project • Choose Tests> Test DevStudio Project • Click the Test Project Developer Studio button 	<ul style="list-style-type: none"> • Choose File> Recent DevStudio Projects • Choose File> Open DevStudio Project • Choose Tests> Test DevStudio Project • Click the Test Project Developer Studio button
C++Test Project	<ul style="list-style-type: none"> • Choose Projects> Open 	<ul style="list-style-type: none"> • Choose Projects> Open

Building a Test Executable

Before C++Test can test a class, it needs to build a test executable that contains all code necessary to call methods from your source file.

C++Test automatically builds a test executable when you click **Test All**.

If you are not going to test a file using **Test All**, you need to prompt C++Test to build a test executable.

To prompt C++Test to build a test executable:

1. In the File List, select the file that you want to test.
2. Click the **Build Test** tool bar button, or choose **Tests> Build Test**.

C++Test will then parse the selected file, read its symbols, instrument it (if necessary), then compile and link it. If you configured C++Test to enforce coding standards with CodeWizard, CodeWizard will test your code at this point.

Initializing Objects

C++Test can automatically initialize the following types of objects:

- Function arguments which are of a complex type. (ARGS)
- Implicit This pointers for member functions. (PreConditions)
- External complex variables used by a given function. (PreConditions)

C++Test will automatically initialize objects when it executes automatically-generated test cases.

When you use user-defined test cases, you can determine which constructor is used to initialize an object. For detailed instructions, see “Initializing Objects with Test Cases” on page 17.

Entering Test Cases

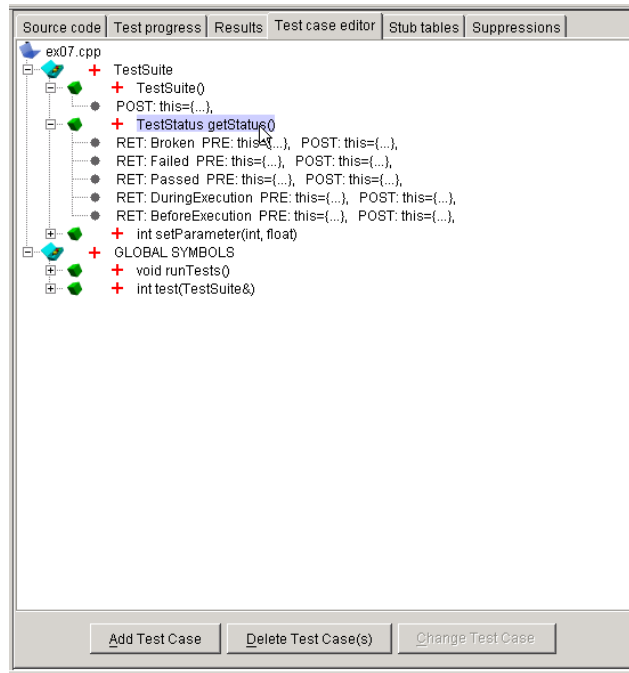
C++Test lets you add your own black-box test cases so you can test your class' functionality. You can add your test cases by specifying arguments, return values, preconditions, and postconditions in the test case skeleton that the Test Case Editor creates for you.

Tip: If you do not enter outcomes, C++Test will automatically determine actual outcomes for the given inputs.

Adding a Test Case

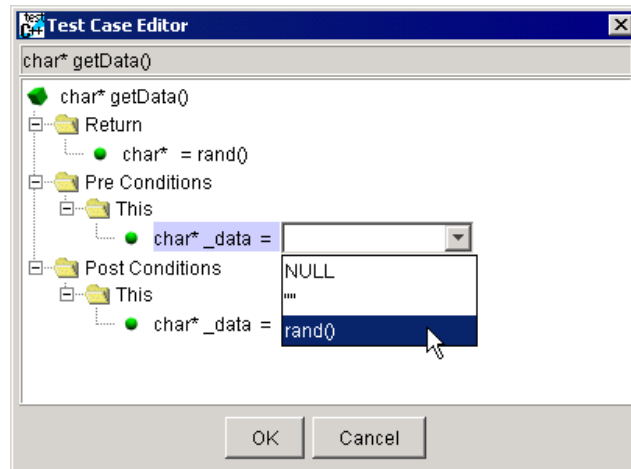
To add a test case:

1. Open the Test Case Editor tab.
2. In the Test Case Editor tab, select the method to which you want to add your first test case, then click **Add Test Case**.



The Test Case Editor window will open.

- In the Test Case Editor window, expand the branch(es) associated with each type of value that you want to enter (for example, Arguments, Return, etc.). You can add values in any node that is marked with a green dot. To add a value, double-click the node to which you want to add a value, then type the value in the text field or choose a predefined value from the drop-down menu.



Important: When dealing with a pointer, you also need to select the allocation size for the pointer element, then expand the pointer to set values for the actual elements.

4. Click **OK**. Your test case will then be added to the Test Case Editor tab's tree.

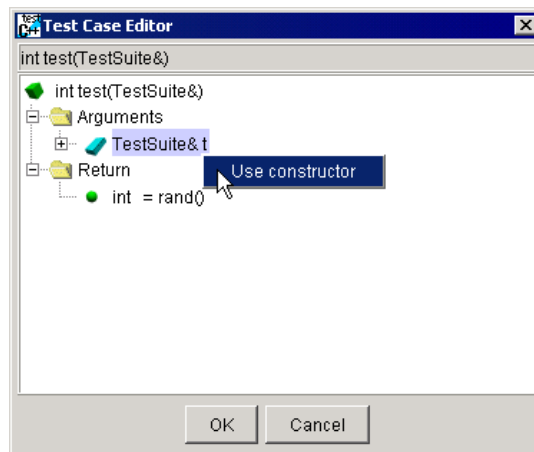
Initializing Objects with Test Cases

C++Test will automatically initialize objects when it executes automatically-generated test cases.

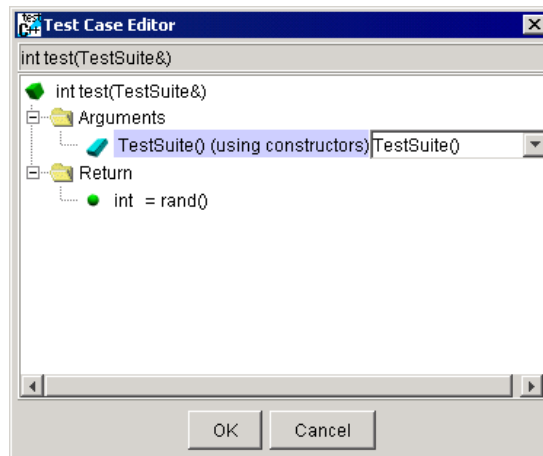
When you use user-defined test cases, you can determine which constructor is used to initialize an object. To do this:

1. Right-click the desired element in the Test Case Editor window.

2. Choose **Use Constructors** from the shortcut menu.



3. Select a constructor from a list of available constructors or type the value that you want to assign to the constructor's argument.



For information on how C++Test initializes objects, see “Initializing Objects” on page 14.

Using 'Char' and String Types in the Test Case Editor Window

You can enter a 'char' value in following formats:

- Single character, captured between apostrophes (e.g. 'a', 'B')
- ESC sequence (e.g. '\n', '\r')
- Character in hexadecimal format (e.g. '\x0ab')
- Character in octagonal format (e.g. '\055')
- Character in numerical format (number from the range 0 - 255 or (-128) - 127)

The character type value is presented in double form: numerical value and the character it corresponds to (e.g. "65 ('A')", "10 ('\n')").

All string types ('char*') values must to be captured between double quotation marks (e.g. "test"), except for the special values NULL / rand().

When these two values are entered without quotation marks, they are treated as a special value; when they are enclosed by quotation marks, they are treated as the actual string.

Editing a Test Case

To edit a test case:

1. In the Test Case Editor tab, select the test case that you want to edit, then click **Change Test Case**.
2. Modify values in the Test Case Editor window that opens.
3. Click **OK**. Your test case will then be modified in the Test Case Editor tab's tree.

Copying and Modifying an Existing Test Case

To copy and modify an existing test case:

1. In the Test Case Editor tab, select the test case that you want to copy.
2. Click **Add Test Case**.
3. Make any desired modifications in the Test Case Editor window that opens.
4. Click **OK**. The new test case will then be added to the Test Case Editor tab's tree.

Removing a Test Case

To remove a test case:

1. Select the test case in the Test Case Editor tab.
2. Click **Delete Test Case** (or right-click the test case and choose **Delete Test Case** from the shortcut menu).

Running a Test

C++Test lets you perform white-box testing, black-box testing, and regression testing. You can perform each of these tests at the file, class, or method level, or you can run single test cases.

General Testing Procedure

Generally, you must open a file and build a test executable for it before you can test the file or any of its classes or methods.

Exception: If you choose **Tests> Test File(s)** or **Tests> Test DevStudio Project**, or if you test a file or DevStudio project from the DevStudio GUI, C++Test will open the file(s), build any necessary test executables, and perform automatic white-box testing in a single step.

The general procedure for testing a file, class, or method is as follows:

1. In the Symbol tree, select the file, class, or method that you want to test.
2. Open the Test Progress tab.
3. Click one of the following buttons:
 - **Record** (to have C++Test automatically generate and execute test cases and execute those automatically-generated test cases).
 - **Play** (to have C++Test execute the test cases that are currently in the Test Case Editor).

For procedures specific to the level of test you are performing, see the following topics:

- “Testing a Method” on page 24
- “Single-Stepping Through a Method” on page 25
- “Testing a Class” on page 26
- “Testing a File, Set of Files, or Developer Studio Project” on page 27
- “Playing a Single Test Case” on page 30

White-Box, Black-Box, and Regression Testing

White-box testing checks that all of a class' methods or functions (including protected and private methods or functions) are robust by determining if the class performs incorrectly when it encounters unexpected input; this type of testing must be performed with full knowledge of the class' implementation details. C++Test performs white-box testing by examining the code, then automatically generating and executing test cases to see how the class behaves when given unexpected input. To perform white-box testing in C++Test, use the **Record** command.

Black-box testing checks a class' functionality by determining whether or not the class' public interface performs according to specification; this type of testing is performed without knowledge of implementation details. C++Test performs black-box testing by testing if the actual outcomes of user-defined and automatically-generated test cases match the expected outcomes (as indicated in the Test Case Editor panel). To perform black-box testing in C++Test, enter your own test cases and/or correct any incorrect outcomes received for automatically-generated test cases, then execute these test cases with the **Play** command.

Regression testing tests whether or not modifications have introduced errors into the class. C++Test performs regression testing by re-executing the same test cases that were run the previous time(s) the class was tested, and checking if any outcomes have changed. To perform regression testing in C++Test, execute all previously run automatically-generated and user-defined test cases with the **Play** command.

Record vs. Play

The **Record** and **Play** commands are both used to run tests and detect failed test cases, exceptions, and other errors. **Record** automatically generates test cases for the selected file, class, or method, then executes the automatically generated test cases. **Play** executes all test cases that are currently available in the Test Case Editor; it does *not* automatically generate any test cases. As explained above, **Record** is typically used to perform white-box testing, while **Play** is typically used to perform black-box or regression testing.

If you want C++Test to automatically generate test cases, you need to use **Record** at least once.

If you want to run test cases that you have added to C++Test, you need to use **Play**.

Generally, it is best to test once using **Record**, add your test cases, then execute both user-defined and automatically generated test cases by clicking **Play**.

Include Files

If the file you that you are testing has an “include” file that is not open in C++Test, you will need to specify where C++Test can find that include file.

To do this:

1. Choose **Options> Project Settings**.
2. In the Project Settings panel’s Build Options tab, choose **Include Options** from the **Build Options** box.
3. Click **Add Includes**, then use the file chooser that opens to indicate what directory contains the appropriate include file.
4. Click **OK** after you have specified the location of your include file(s).

Template Support

C++Test can test instantiated template functions and any non-template function that takes an instantiated template as an argument or returns a template instance.

Testing a Method

There are two ways to test a single method:

- From the Test Progress tab
- From shortcut menus

Both ways require that you first build a test executable.

From the Test Progress Tab

To test a method from the Test Progress tab:

1. In the Symbol tree, select the method that you want to test.
2. Open the Test Progress tab.
3. Click one of the following buttons:
 - **Record** (to have C++Test automatically generate test cases and execute those automatically-generated test cases).
 - **Play** (to have C++Test execute the test cases that are currently in the Test Case Editor).

From Shortcut Menus

To test a method from any tree's or panel's shortcut menu:

1. In any C++Test tree or panel, right-click the method that you want to test.
2. Choose one of the following commands from the shortcut menu:
 - **Record Tests** (to have C++Test automatically generate test cases and execute those automatically-generated test cases).
 - **Playback Tests** (to have C++Test execute the test cases that are currently in the Test Case Editor).

Single-Stepping Through a Method

When you test a single method from the Test Progress tab, you can attach a debugger and single-step through the method.

To single-step through a method:

1. In the Symbol tree, select the method that you want to single-step through.
2. Open the Test Progress tab.
3. Check **Attach debugger**.
4. Click either **Record** (to have C++Test automatically generate test cases and execute those automatically-generated test cases), or **Play** (to have C++Test execute the test cases that are currently in the Test Case Editor).

As C++Test starts testing, DevStudio will open.

5. In DevStudio, select the appropriate file from the dialog box that opens, then click **OK**.
6. Click DevStudio's **Break Execution** button.
7. Click the **F5** (Refresh) button.

The debugger will break at the beginning of the method.

8. Single-step through the code as normal.

Testing a Class

There are two ways to test a class:

- From the Test Progress tab
- From shortcut menus

Both ways require that you first build a test executable.

From the Test Progress Tab

To test a class from the Test Progress tab:

1. In the Symbol tree, select the class that you want to test.
2. Open the Test Progress tab.
3. Click one of the following buttons:
 - **Record** (to have C++Test automatically generate test cases and execute those automatically-generated test cases).
 - **Play** (to have C++Test execute the test cases that are currently in the Test Case Editor).

From Shortcut Menus

To test a class from any tree's or panel's shortcut menu:

1. In any C++Test tree or panel, right-click the class that you want to test.
2. Choose one of the following commands from the shortcut menu:
 - **Record Tests** (to have C++Test automatically generate test cases and execute those automatically-generated test cases).
 - **Playback Tests** (to have C++Test execute the test cases that are currently in the Test Case Editor).

Testing a File, Set of Files, or Developer Studio Project

There are a variety of different ways to test a file, set of files, or a DevStudio project. Some ways require that you open the file and build a test executable prior to starting the test; others will automatically open files, build test executables, and start the test in a single step.

Automatically Testing Unopened Files

If you do not have any files open and you are not going to make any modifications to test parameters, the fastest way to start a test is to perform any of the following procedures:

- Click the Developer Studio **Test File** button (to test a selected file in your Developer Studio workspace).
- Click the Developer Studio **Test Project** button (to test the entire project in your Developer Studio workspace).
- Choose **Tests> Test File(s)** (to select and test files from the C++Test GUI).
- Choose **Tests> Test DevStudio Project** (to select and test a DevStudio project from the C++Test GUI).

Any of these commands will prompt C++Test to perform the following actions:

- Open (or let you select) the file or Developer Studio project that you want to test.
- Build any necessary test executables.
- Automatically generate and execute white-box test cases.

Automatically Testing Opened Files

If the file(s) that you want to test is open, does not yet have a test executable, and you are not going to make any further modifications to test

parameters or add test cases, the fastest way to test it is to click **Test All** or choose **Test> Test All**. C++Test will then build an executable for all files currently in the File List, automatically generate test cases for all of these files, then execute all automatically-generated test cases.

Testing a Single File

If you want to test a single file, there are two additional ways you can start the test:

- From the Test Progress tab
- From shortcut menus

Both ways require that you first build a test executable.

From the Test Progress Tab

To test a file from the Test Progress tab:

1. In the Symbol tree, select the file that you want to test.
2. Open the Test Progress tab.
3. Click one of the following buttons:
 - **Record** (to have C++Test automatically generate test cases and execute those automatically-generated test cases).
 - **Play** (to have C++Test execute the test cases that are currently in the Test Case Editor).

From Shortcut Menus

To test a file from any tree's or panel's shortcut menu:

1. In any C++Test tree or panel, right-click the file that you want to test.
2. Choose one of the following commands from the shortcut menu:
 - **Record Tests** (to have C++Test automatically generate test cases and execute those automatically-generated test cases).

- **Playback Tests** (to have C++Test execute the test cases that are currently in the Test Case Editor).

Playing a Single Test Case

There are two ways to play a single test case:

- From the Test Case Editor tab
- From the Results tab

Both ways require that you first build a test executable.

From the Test Case Editor Tab

To play a test case from the Test Case Editor tab:

1. Open the Test Case Editor tab.
2. Right-click the test case that you want to play, then choose **Play Test Case** from the shortcut menu.

From the Results Tab

If you have already run a test and have results in your Results tab, you can play a test case from the Results tab. To play a test case from the Results tab:

1. Open the Results tab.
2. Right-click the test case that you want to play, then choose **Play Test Case** from the shortcut menu.

Test Results and Output

C++Test collects test results in several areas and formats:

- **The Results tab:** A graphical tree that provides a summary of test results.
- **The Test Result Viewer window:** A graphical tree that provides detailed test results for one of the test cases represented in the Results tab.
- **The Source Code tab:** A source code viewer that indicates at what line of the source code each problem occurs.
- **The Messages tab:** A tab that guides you through the errors found. It also lets you open detailed test case results in the Test Result Viewer window and the source file in the editor of your choice.
- **Output:** Contains text formatted data.

You can get information about results by:

- Viewing a summary of all test results.
- Viewing details about a specific test case.
- Viewing details about errors found.
- Viewing text-format output.

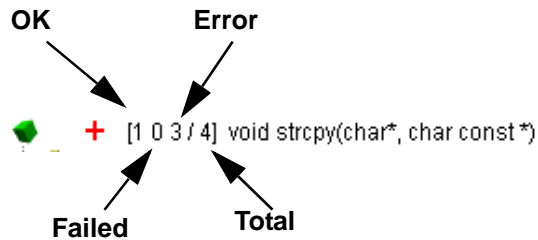
Important: If you configured C++Test to use CodeWizard and/or Insure++, you can access the messages from these tools in the Source Code tab and the Messages tab. For information on accessing CodeWizard/Insure++ test information, see “Viewing Details About Errors Found” below.

Viewing a Summary of All Test Results

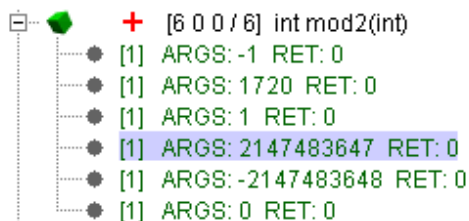
You can see the following test summary information in the Results tab:

- The total number of test cases used

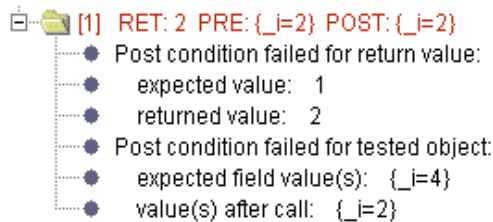
- A summary of how many tests are in each of the three available result categories:
 - **OK:** The method returned and had the correct return value and/or postcondition
 - **Failed:** The test did not have the correct return value or postcondition
 - **Error:** The method crashed



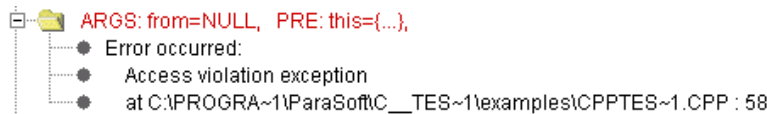
- Results for each test case
 - If a test case is OK, expand its branch to see (in green) the number of times it was executed (in brackets) and its arguments, returns, preconditions, and postconditions.



- If a test case failed, expand its branch to see the number of times it was executed (in brackets, in red), its arguments, returns, preconditions, and postconditions (in red), and the reason why it failed.



- If a test case had an error, expand its branch to see the number of times it was executed (in brackets, in red), its arguments, returns, preconditions, and postconditions (in red), and details about the type of exception found and at what line number the error occurred.



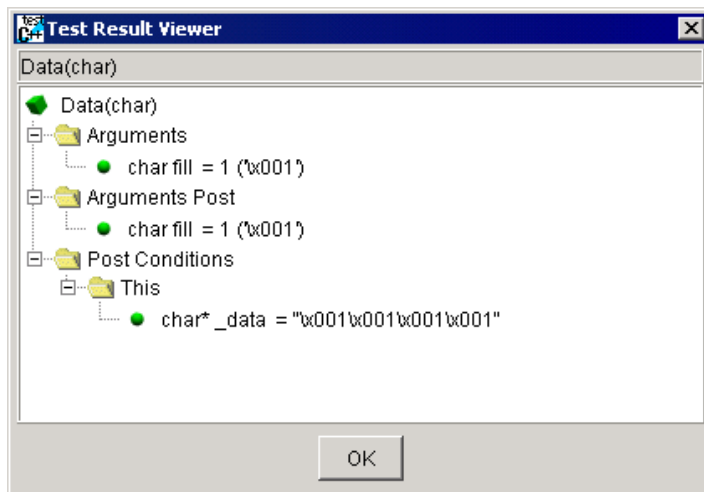
Controlling Which Results are Displayed

You can control what results are displayed in the Results tab by clicking items in the Symbol tree. For example, to see all results for a file, you would click the file name. To see results for a single class, click that class. To see results for a certain method, click that method.

By default, all results for the selected file, class, or method are displayed in the Results tab. If you only want to see the most current results, choose **last results** from the **Show** box at the top of the tab. If you want to focus on OK test cases, test cases with errors, or failed test cases, choose the appropriate category from the **of type** box at the top of the tab.

Viewing Details About a Specific Test Case

To view details about any test case in the Results tab, right-click that test case, then choose **View Test Results** from the shortcut menu that opens. The Test Result Viewer window will then open and contain detailed results about the selected test case.



Test case information is organized into the following categories:

- **Arguments:** Values in this branch represent the function's arguments.
- **Arguments Post:** Values in this branch represent the function's arguments after the test.
- **Return:** Values in this branch represent the function's result.
- **Pre Conditions> This:** Values in this branch represent the condition of the object before the test.
- **Pre Conditions> Externals:** Values in this branch represent the condition of the global variables before the test.

- **Post Conditions> This:** Values in this branch represent the condition of the object after the test.
- **Post Conditions> Externals:** Values in this branch represent the condition of the global variables after the test.

Each category of information is represented by a branch in the Test Result Viewer window. The branches used depend upon the function under test; not all test cases will contain all of the above branches.

Viewing Details About Errors Found

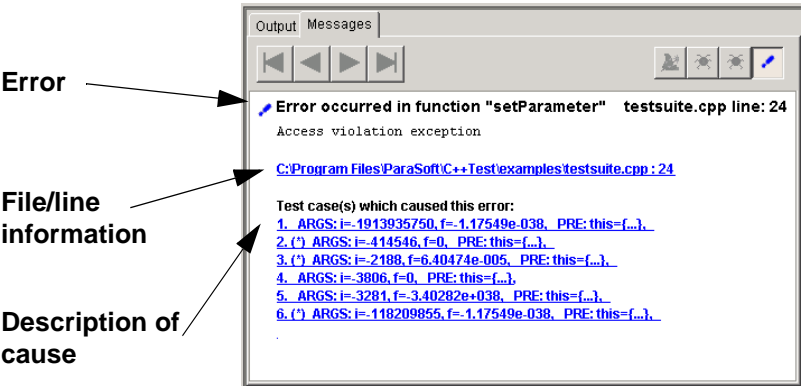
The Messages tab provides detailed information about each error found by C++Test, CodeWizard (if applicable) and Insure++ (if applicable).

There are two ways to open the Messages tab:

- Select the red Results tree node that represents a test case that uncovered an error.
- Open the Source Code tab after a test has been completed.

Each message displayed in the Messages tab contains the following items:

- Type of error. If the error found is an exception, this node will display the type of exception that occurred (for example, "Access violation exception" or "Integer divide by zero exception").
- File and line number where the error occurred. You can open the source in the integrated editor by clicking this link.
- Description of what caused the error. When the error was found via dynamic analysis, this description is a list of test cases that caused the error. You can open a test case in the Test Result Viewer by clicking its link. You can also run the test case by right-clicking its link and choosing **Play Test Case** from the shortcut menu.

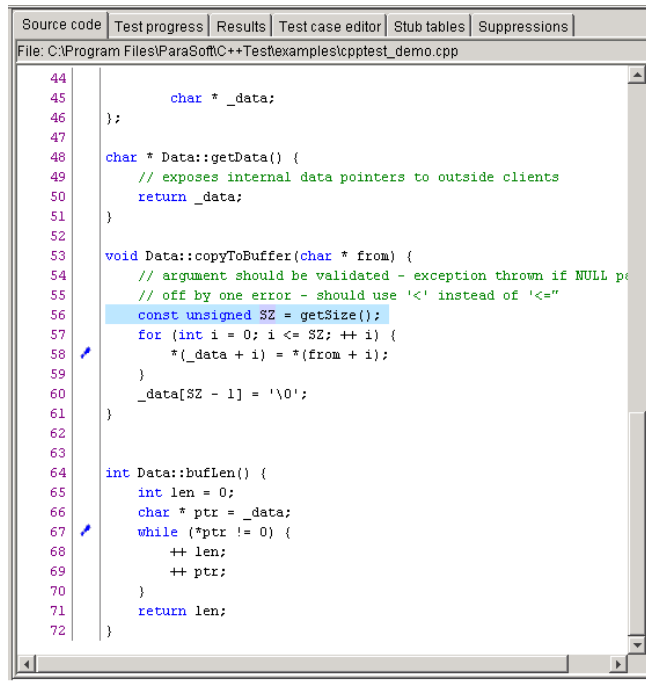


You can scroll through all available messages using the navigation buttons at the top left of the Messages tab. Available buttons include:


⏪	Go to the first message.
⏴	Go to the previous message.
⏵	Go to the next message.
⏩	Go to the last message.




The Messages tab works with the Source Code tab to help you see exactly where the errors/violations that C++Test, CodeWizard (if applica-

ble), and Insure++ (if applicable) report occur. When a message is selected in the Messages tab, the Source Code tab will highlight the related line of source code. When a problem has been detected at a certain line of source code, C++Test places the appropriate icon to the left of that line of source code.



Icons used in the Source Code tab include:

Icon	Description	Indicates
	Red wizard hat	Violation CodeWizard found when statically analyzing the code.

	Red bug	Error Insure++ found at compile-time.
	Blue bug	Error Insure++ found at runtime.
	Blue exclamation point	Error C++Test found when running the test.

If you want to view the message that an icon represents, click that icon. Information about the selected message will be displayed in the Messages tab.

Determining What Types of Messages Are Displayed

To determine what types of messages are displayed in the Source Code tab and the Messages tab, toggle error types on and off using the icon buttons in the top right of the Messages tab. Suppressed messages will be indicated with gray icons in the Source Code tab.

Viewing Text-Format Output

You can see the following output in the Output tab as or after you run a test:

- File name
- Class name
- Test number (each test is automatically numbered by C++Test)
- Preconditions
- Arguments
- Return values
- Postconditions

- Status:
 - **OK:** The method returned and had the correct return value and/or postcondition
 - **Error:** The method crashed
 - **Failed:** The test did not have the correct return value or postcondition

Working With Results and Output

Several actions that you might want to perform when viewing results and/or output include:

- **Edit source code:** To edit the source code responsible for a certain error, click the file/line information in the Messages tab.
- **Print your results:** To print your results, choose **File> Print Report** or **File> Print Preview**.
- **Save your output:** To save your output in text format, right-click any area of the Output tab, choose **Save Output** from the shortcut menu, then specify where you want to save the output.
- **Replay a test case:** To replay a single test case, right-click that test case in the Result tree, then choose **Play Test Case** from the shortcut menu. Or, right-click that test case in the Messages window, then choose **Play Test Case** from the shortcut menu.
- **Edit a test case:** To edit a certain test case, right-click that test case in the Result tree and choose **View Test Case** from the shortcut menu. (Or, click the test case in the Messages window). The test case will then open in the Test Case Editor. You can then edit your test case in the normal manner.
- **Edit stubs used in a test case:** To edit the stubs used in a certain test case, right-click that test case in the Result tree and choose **View Stub Configuration** from the shortcut menu. The stubs used in the selected test case will then open in the Stub Tables tab. You can then edit stubs in the normal manner.
- **View coverage information:** See “Monitoring Test Coverage” on page 56.

- **Prevent C++Test from testing a class or method in future tests:** See “Excluding Specific Classes or Methods From Your Test” on page 51.

Validating Test Case Outcomes

In order to ensure the accuracy of future tests, you must validate test case outcomes. Validating test case outcomes involves determining if certain arguments and preconditions led to expected returns and postconditions.

You do not need to validate outcomes for test cases where you entered your own preconditions, arguments, returns, and postconditions because you have already indicated the correct input/outcome relation.

However, you do need to validate the output of test cases for which C++Test automatically generated outcomes. When C++Test automatically generates outcomes, it displays actual outcomes, not expected outcomes. If an outcome is incorrect, subsequent tests will report this same outcome as correct, and might report errors if your outcome is *actually* correct. Thus, it is important to indicate the correct outcome.

To validate outcomes:

- If an outcome is correct, no action is required.
- If an outcome is incorrect, you need to indicate the correct outcome value. To indicate the correct outcome value:
 1. Go to the Test Case Editor tab, select the test case whose outcome you want to correct, then click **Edit Test Case**. The Test Case Editor window will open.
 2. In the Test Case Editor window, enter the correct return and/or postcondition values.
 3. Click **OK** to close the Test Case Editor window and modify the test case.

The modified values will then be saved and used as reference values for all subsequent tests.

Editing Source Code

If you want to change your source code within the C++Test environment, you can do so in your preferred source code editor.

To edit a file's source code, go to the File List and select the file you want to edit, then choose **File> Edit Source**. The file will open in Notepad, or in the source code editor that you specified in the Customize panel.

Changing Your Source Code Editor

To change what editor is opened when you choose **File> Edit Source**:

1. Choose **Options> Customize**. The Customize panel will open.
2. In the Editors area of the Customize panel's General Options tab, select your preferred editor.
 - To choose Developer Studio, click **DevStudio**.
 - To choose Notepad (the default setting), click **Notepad**.
 - To select an editor that is not listed, click **Other** and enter the full path to the editor's executable, followed by any arguments that you want to pass to the editor, in the field to the right of the **Other** option.
3. Click **OK** to close the Customize panel.

Viewing Source Code

To view a file's source code:

1. Click the Source Code tab.
2. In the File List, select the file whose source code you want to view.

The selected file's source code will then be displayed in the Source Code tab's source viewer.

Viewing Specific Methods

If you have a large file, you may only want to look at the source code of a specific method. You can do this after C++Test has parsed your code (code is parsed when C++Test reads symbols or builds a test executable).

To have the source code for a single method highlighted in the source viewer:

1. Click the Source Code tab.
2. In the Symbol tree, select the method that you want highlighted in the Source Viewer.

The selected method will then be highlighted in the source viewer.

Customizing the Source Viewer

You can tailor the source viewer's properties to reflect the conventions that you are used to or would prefer. You can change what colors are used to signify various elements as well as determine whether or not line numbers are displayed. To customize the source viewer's properties:

1. Right-click any area of the Source Code tab.
2. Choose **Text Properties** from the shortcut menu that opens. The Text Properties panel will open.
3. In the Text Properties panel, make any desired modifications. Available options include:

- **Show line numbers:** Determines whether line numbers are displayed.
 - **Color code syntax:** Determines whether color is used to indicate the code's syntax.
 - **Element Color:** Determines the color for each syntactical element. You can change the color used for an element by selecting the element from the list of elements, using the RGB sliders to choose a new color, then clicking **Set Color**.
 - **Font:** Determines what font face, size, and type are used. You can change the font by selecting the desired settings from the available boxes, then clicking **Set Font**.
4. Click **OK** to close the Text Properties window and apply your changes.

Your changes will then be applied to the source viewer and will be used for all subsequent tests.

Refreshing Your Symbol Tree to Reflect Code Modifications

C++Test's Symbol tree is based on your source at the last time that you read symbols or built an executable. If you have made changes to your file that affect the Symbol tree, you must update the symbols database and Symbol tree to reflect those changes. If your Symbol tree does not match your actual tree, your tests will not be accurate.

To update your symbols database and Symbol tree:

1. Go to the File List and select the file whose symbols you want to update.
2. Click one of the following buttons:
 - **Read Symbols:** Click this button if you do not want to rebuild your file at this point. If your Symbol tree has changed, you will need to build a new test executable before you run a test.
 - **Build Test:** Click this button to recreate the Symbol tree and rebuild your test executable.

Performing Regression Testing

Each time that you modify a class, you should perform regression testing to ensure that your changes have not introduced errors.

C++Test completely automates all steps involved in and related to regression testing. Whenever C++Test tests a class, it saves the tests and test parameters. When you are ready to perform regression testing, you can run all previous white-box and (if available) black-box test cases by:

1. Opening the appropriate file.
2. Building a test executable.
3. Selecting the file, class, or method that you want to test.
4. Clicking the Test Progress tab's **Play** button.

C++Test then automatically runs the same test cases that it ran during the previous test, with the same test parameters. Any problems found are displayed in the Results tab.

Working With Stubs

C++Test uses stub functions when a method being tested calls an external function that is not available or accessible. Rather than actually calling the function, C++Test calls the stub and uses the return values that the stub provides.

C++Test always creates stubs when the method under test calls an external function that is unavailable or inaccessible. When a C++Test generated stub is called, it returns 0 or NULL by default; if you want to control what return values are used, you can create a stub table that specifies what outcome should be used for certain input.

If C++Test creates a stub for an external function but you would rather use the actual external function, you can have C++Test use that original function by changing the stub type (as described below) and telling C++Test where the original function is located.

You can also enter user-defined stubs. You should use C++Test generated stubs when you do not care what that method does, or when you want to use it as a transformer for a particular input without recompilation and slowing down the test execution. However, C++Test generated stubs have very limited capabilities, and you might want to use a user-defined stub if you want to simulate the behavior of the original function by replacing the original function with a simplified one.

The main drawback of using a user-defined stub is that it requires recompilation of the test executable and might slow down the test execution.

Configuring Stubs

In the Stub Tables tab, you can view and determine what type of stub is used, enter user-defined stubs, and create stub tables.

Viewing/Modifying Stub Types

1. In the Symbol tree, select the file, class, or method whose stubs you want to modify.

2. Click the Stub Tables tab. A list of all the selected [in the Symbol tree] file's, class', or method's functions and each function's current stub types will be displayed in this tab. If C++Test sees the function, it will use the original function by default. If it does not, it will use a C++Test generated stub.
3. To change a stub type, click the appropriate button at the bottom of the Stub Tables tab.

Note: If you change from a C++Test-generated stub to an original function, C++Test will open a dialog box that lets you specify the location of the external function's library.

Warning: Substituting an original function for a C++Test-generated function can cause link failures when building a test executable.

Defining Your Own Stubs

1. Double-click the appropriate row in the **Function** column.

An editor will open with an empty stub file (the editor used will vary depending on your settings). The empty stub file will look something like this:

```
/*
 * DO NOT CHANGE THE SIGNATURE OF THIS METHOD - JUST FILL ITS BODY
 * User stub for:
 *     bool odd(int)
 */
bool _MTU_odd(int)
{
}
```

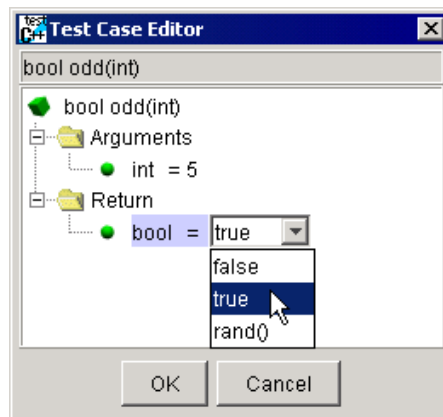
2. Add code to the function. You can provide names for the parameters (if you intend to use them). Do not change the name or signature of the function, and always return the proper type.
3. Save the code in your editor.
4. In the bottom of the Stub Tables tab, click the **User Defined** option.

Specifying Return Values for C++Test Generated Stubs

When you are using C++Test generated test cases, you can specify return values and postconditions for certain arguments and preconditions by creating a stub table with test cases. Whenever the method is called and the input values match the ones in the stub table, the outcome values will match the ones in the stub table. If the input values do not match the ones in the stub table, the return will be 0 or NULL (depending on the return type) as long as the argument is not void, and the postconditions will be the same as the preconditions.

To specify return values by entering test cases:

1. In the Stub Tables tab, select the C++Test generated stub whose return values you want to specify, then click **Add Entry**. The Test Case Editor window will open.
2. In the Test Case Editor window, specify input values and corresponding outcome values by entering test cases. Enter these test cases the same way that you enter test cases in the Test Case Editor.



C++Test will use these input/outcome correlations when it tests the related function.

Excluding Specific Classes or Methods From Your Test

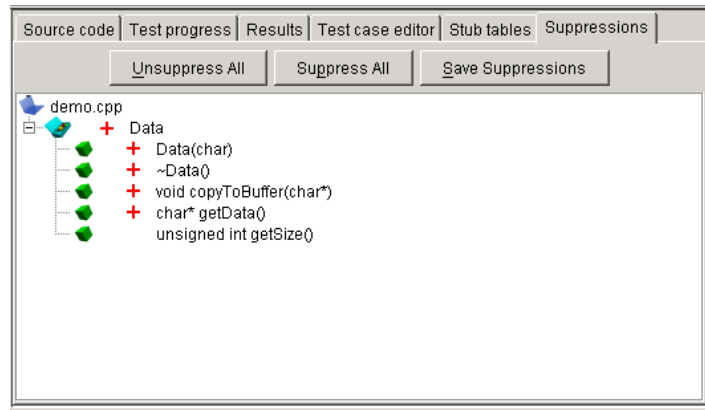
Sometimes you may not want to test every class or method in your file or project. You can tailor C++Test to test only the classes and methods that you want tested by suppressing those that you do not want tested. You can suppress classes or methods in two ways:

- Using the Suppressions Tab
- Using Shortcut Menus

Using the Suppressions Tab

In the Suppressions tab (as well as in other C++Test tabs), a red plus sign indicates that a class or method will be tested. As in other tabs, you can control what classes or methods appear in the tab by selecting a file, class, or method in the Symbol tree.

To prevent a class or method from being tested, remove the plus sign in the Suppressions Tab. You can remove a plus sign by simply clicking it; if you later want to unsuppress the class or method, click the area where the plus sign should appear and the plus sign will reappear.



To suppress or unsuppress all methods and classes that appear in the Suppressions tab, click **Suppress All** or **Unsuppress All**.

You can save your suppression scheme for future tests or to share with other members of your team. To save your suppression scheme, click **Save Suppressions**.

Using Shortcut Menus

You can also suppress a class or method by right-clicking its node or table row in the Symbol tree, Results tab, Test Case Editor tab, or Stub Tables tab, then choosing **Suppress Class** or **Suppress Method** from the shortcut menu that opens.

If you want to unsuppress a class or method, you must do so in the Suppressions tab.

Enforcing Static Coding Standards With CodeWizard

You can prevent errors by using CodeWizard to automatically enforce industry-wide and custom static coding standards-- rules designed to reduce the possibility of introducing errors, as well as increase code portability and maintainability. Once configured to do so, C++Test will automatically run each file through CodeWizard when it reads a file's symbols, then it will alert you to any violations.

Configuring CodeWizard

In order to use CodeWizard, you must have CodeWizard (with a valid CodeWizard license) installed on your machine.

To configure C++Test to automatically run your classes and methods through CodeWizard, enable the **Use CodeWizard** option by choosing **Options> Project Settings**, then selecting the **Use CodeWizard** option in the Build Options tab.

If you want to modify any CodeWizard settings, you can invoke the CodeWizard Control Panel by clicking **CodeWizard Panel** in that same Project Settings tab. Information on the available settings can be found in your CodeWizard User's Guide.

Running CodeWizard

When the **Use CodeWizard** option is enabled, CodeWizard will automatically enforce coding standards when it reads a file's symbols. (Symbols are read when you click **Read Symbols**, **Build Test**, or **Test All**).

If you enable the **Use CodeWizard** option *after* C++Test has read a file's symbols, C++Test will run CodeWizard the next time symbols are read.

CodeWizard violations will be reported in the Source Code tab and the Messages tab. For information about viewing results, see "Test Results and Output" on page 31. For information about specific CodeWizard violations, refer to your CodeWizard User's Guide.

Detecting Runtime Errors With Insure++

If you want to perform runtime error detection on the classes and methods that you test with C++Test, you can configure C++Test to automatically run the classes and methods through Insure++ as they are being tested by C++Test. You will then be able to automatically detect the following types of errors at the unit level:

- Memory corruption/uninitialized memory
- Memory leaks
- Memory allocation errors
- Variable definition conflicts
- I/O errors
- Pointer errors
- Library errors
- Logic errors

Configuring Insure++

In order to use Insure++, you must have Insure++ (with a valid Insure++ license) installed on your machine.

To configure C++Test to automatically run your classes and methods through Insure++ by choosing **Options> Project Settings**, then selecting the **Use Insure++** option in the Build Options tab.

If you want to modify any Insure++ settings, you can invoke the Insure++ Control Panel by clicking **Insure++ Panel** in that same Project Settings tab. Information on the available settings can be found in your Insure++ User's Guide.

Running Insure++

When the **Use Insure++** option is enabled, Insure++ will statically analyze your code when you build a test, then dynamically analyze your code during the test.

If you enable the **Use Insure++** option *after* you have built a test executable, C++Test will instrument the class with Insure++ when you start a test.

Errors found by Insure++ will be reported in the Source Code tab and the Messages tab. For information about viewing results, see “Test Results and Output” on page 31. For information about specific errors Insure++ exposes, refer to your Insure++ User’s Guide.

Monitoring Test Coverage

If you want to monitor coverage, you can view real-time coverage information, the number of times each line was executed, and a summary of total coverage. This information will help you gauge the effectiveness of your current test suite and give you the information that you need to achieve the greatest possible coverage.

Configuring C++Test to Monitor Coverage

C++Test does not monitor coverage by default. To configure C++Test to monitor coverage, enable coverage by choosing **Tests> Enable Coverage**. This will prompt C++Test to monitor coverage for all subsequent tests.

To stop C++Test from monitoring coverage, disable coverage by choosing **Tests> Enable Coverage**.

Monitoring Coverage During a Test

To see a real-time depiction of how C++Test is covering your file, class, or method, you can view coverage information as a test is being performed. To view real-time coverage information:

1. Enable coverage (as described above).
2. As the test is being performed, view the coverage information in the Coverage window.
 - A red line indicates the line currently being executed.
 - A green line indicates lines that have been covered.
 - The number to the left of the line of code indicates the number of times that line was executed.

Line that was
already covered

Number of times
line was executed

Line currently
being executed

```

20  }
21  }
22  // correct version of bubble sort
23  void bubble_sort2(int* arr, int size)
24  {
25      for (int i=0; i < size; ++i) {
26          for (int j=0; j < size-i-1; ++j) {
27              if (arr[j] > arr[j+1]) {
28                  int temp = arr[j+1];
29                  arr[j+1] = arr[j];
30                  arr[j] = temp;
31              }
32          }
33      }
34  }
35  }
36  // this function has an overwrite error
37  .

```

Viewing a Coverage Summary

To view a summary of all coverage since the first test in which coverage was enabled (or the last time you cleared coverage information):

1. Open the Coverage window by choosing **Tests> Show Coverage**.
2. View the coverage information in the Coverage window.
 - A blue line indicates lines covered since the first time the file was tested after coverage was enabled, or since the coverage information was cleared. (For information on clearing coverage, see Clearing Coverage below).
 - The number to the left of the line of code indicates the number of times that line was executed since the first time the file was tested after coverage was enabled, or since the coverage information was cleared.

Line that was covered

Number of times line was executed

```

20      }
21      }
22      }
23      // correct version of bubble sort
24      void bubble_sort2(int* arr, int size)
25      {
26          for (int i=0; i < size; ++i) {
27              for (int j=0; j < size-i-1; ++j) {
28                  if (arr[j] > arr[j+1]) {
29                      int temp = arr[j+1];
30                      arr[j+1] = arr[j];
31                      arr[j] = temp;
32                  }
33              }
34          }
35      }
36      }
37      // this function has an overwrite error

```

Clearing Coverage

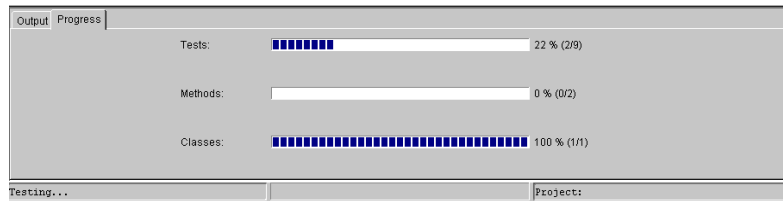
C++Test keeps track of and displays all coverage information gathered since the first time a file was tested or since the last time coverage was cleared. To clear the coverage, choose **Tests> Clear Coverage**.

Monitoring Test Progress

As C++Test performs tests, you can monitor test progress in the Progress tab at the bottom of the GUI.

This tab contains information about the number and percentage of:

- Tests performed
- Methods tested (if you are testing a class or file)
- Classes tested (if you are testing a file)



Customizing General Test Settings

General test settings apply to general C++Test operations; they are not associated with a specific project, file, class, or function. General test settings can be configured in the Customize panel. You can reach this panel by choosing **Options> Customize**.

To modify general test settings:

1. Open the Customize panel by choosing **Options> Customize**.
2. Modify any settings that you want to change.
3. When you are done modifying settings, click **OK**.

General test settings are organized into two tabs: the General Options tab and the Text Properties tab.

General Options Tab

The General Options tab includes the following options:

- **Editors:** Select your preferred editor. If you want to select an editor that is not listed, click **Other** and enter the full path to the editor's executable, followed by any arguments you want to pass to the editor, in the field to the right of the **Other** option.
- **Start C++Test With:** Allows you to determine what (if any) C++Test Project is automatically loaded into the C++Test GUI upon startup.
- **Suppressions:** Allows you to determine whether suppressed symbols are displayed in the Symbol Tree. Hiding suppressed symbols in the Symbol Tree can make that tree easier to analyze and manage.
- **Default Initial Suppress Mode:** Allows you to determine whether C++Test automatically suppresses symbols from headers files.

Text Properties Tab

The Text Properties tab includes the following options:

- **Show Line Numbers:** Determines whether line numbers are displayed in the Source Viewer
- **Color Code Syntax:** Determines whether color is used to indicate the code's syntax in the Source Viewer.
- **Element Color:** Determines the color for each syntactical element in the Source Viewer. You can change the color used for an element by selecting the element from the list of elements, using the RGB sliders to choose a new color, then clicking **Set Color**. To restore default colors, click **Default Colors**.
- **Font:** Determines what font face, size, and type are used in the Source Viewer. You can change the font by selecting the desired settings from the available boxes, then clicking **Set Font**.

Customizing Project Test Settings

Project test settings apply to all C++Test tests. Project test settings can be configured in the Project Test Settings panel. You can reach this panel by choosing **Options> Project Settings**.

To modify project test settings:

1. Open the appropriate Settings panel by choosing **Options> Project Settings**.
2. Modify any settings that you want to change.
3. When you are done modifying settings, click **OK**.

Project test settings are organized into two tabs: the Build Options tab and the Ignore Properties tab.

Build Options Tab

The Build Options tab includes the following options:

- **Compiler:** Specifies what compiler you want C++Test to use.
- **Build Options:** Determines how C++Test builds your test executables. This box includes the following options:
 - Preprocessor flags.
 - Include options: Click the **Add Includes** button to open a file chooser where you can specify the location of your include files.
 - Compiler options.
- **CodeWizard options:** The **Use CodeWizard** option allows you to determine whether C++Test uses CodeWizard to enforce coding standards during its tests. The **CodeWizard Panel** button opens the CodeWizard Control Panel that allows you to set CodeWizard configuration options.

- **Insure++ options:** The **Use Insure++** option allows you to determine whether C++Test uses Insure++ to find compile-time and runtime errors during its tests. The **Insure++ Panel** button opens the Insure++ Control Panel that allows you to set Insure++ configuration options.

Ignore Paths Tab

The Ignore Paths tab specifies which paths/files you do not want C++Test to test. To add a path to this list either:

- Click the ellipses button, then use the file chooser to specify a path.
- Enter the path in the text field at the bottom of the Ignore Paths tab, then click **Add**.

To restore the default "ignore path" settings, click **Defaults**.

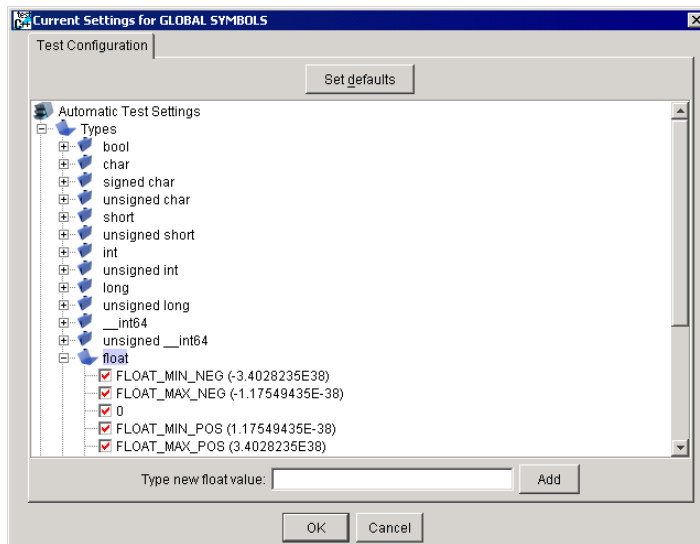
Customizing Automatic Test Settings

You can control what types of white-box test cases are generated by customizing automatic test settings.

When you customize automatic test settings, you can determine what kinds of arguments are used to test each type, as well as specify general test settings such as number of test cases, timeout time, and how deeply C++Test searches embedded classes. Automatic test settings can be configured at the file, class, or function level in the appropriate Current Settings panel. You can reach this panel by selecting the file, class, or function whose automatic test settings you want to modify, then choosing **Options> Test Configuration**.

To customize automatic test settings:

1. In the Symbol tree, select the file, class, or function whose settings you want to customize, then choose **Options> Test Configuration**. For example, to open a class' test settings, select the node that represents that class in the Symbol tree, then choose **Options> Test Configuration**. The appropriate Current Settings panel will open.
2. In the Current Settings panel's Test Configuration tab, specify what arguments you want to use for each type.
 - To prevent an argument from ever being used, clear that argument's check box.
 - To make an argument available for C++Test's automatically generated test cases, check that argument's check box. The argument will only be used when it applies to the method under test.
 - To add a new argument, select the type to which that you want to add the argument, enter the argument value in the **Type new <type> value** field, then click **Add**.



3. Modify any test generation settings that you want to change. Available options include:
 - **Max. Count of Generated Test Cases:** The maximum number of test cases that C++Test will generate for a single test. Any integer greater than zero is allowed.
 - **Max Depth of Field Inspection:** The maximum depth of nested classes for which C++Test will generate test cases. Any integer greater than zero is allowed.
 - **Timeout (in seconds) for Test Case Execution:** The maximum number of seconds that C++Test will wait for a test case to execute. After this amount of time has elapsed, C++Test will move on to the next test case, or it will stop testing (if all other test cases have been executed). Any integer greater than zero is allowed.

- **Max. Dynamic Allocation Size:** The maximum number of bytes that can be used when dynamically allocating memory. This parameter is used to prevent C++Test from allocating excessive amounts of memory to data types that need dynamic memory allocation. Any integer greater than zero is allowed.

4. When you are done modifying test settings, click **OK**.

Your settings will then be applied and used during subsequent tests.

To restore all settings to their default values, click **Set Defaults**.

Working With C++Test Projects

Whenever C++Test performs a test, it automatically saves the test parameters and test cases (both automatically generated and user-defined). When you open a previously tested file, its parameters are automatically restored. This lets you accurately repeat a test to see if class modifications caused class behavior to change and/or introduced errors (i.e. it lets you perform regression testing).

If you frequently work with a certain set of files or want to easily apply a set of parameters to files, you should create and save a C++Test Project. C++Test Projects contain all of the files and parameters for a certain set of files; when you open a C++Test Project, all of the project's files and parameters are restored.

Creating a C++Test Project

To create a C++Test Project:

1. Choose **Project> New Project**, and enter a project name in the dialog box that appears.
2. Open the files that you want to include in the project.
3. Save the project by choosing **Project> Save Project**.

When you save a C++Test Project, C++Test creates a .ctp file for this project in <C++Test_install_dir>/C++TestProj/<username>/Projects.

Saving a C++Test Project

To save a C++Test project, choose **Project> Save Project** or **Project> Save Project As**.

Restoring a C++Test Project

To restore a project:

1. Choose **Project> Open Project**.
2. Choose a project in the dialog box that opens.
3. Click **OK**.

The project's files and test parameters will then be restored.

Deleting a C++Test Project

To remove a C++Test Project, restore it, then choose **Project> Delete**.

Configuring C++Test to Open a Project on Start Up

By default, C++Test does not load any projects in its GUI upon start up. You can configure C++Test to open either a specific project or the last opened project when it is launched. To do this:

1. Choose **Options> Customize**.
2. Determine what project you want opened upon startup:
 - If you want C++Test to open the last opened project, select the **Last Opened Project** option.
 - If you want C++Test to open a specific project, click the ellipses (...) button, then choose the name of the project that you want opened.
3. Click **OK**.

Sharing Tests

To share your test with another user, create a C++Test project that includes the test that you want to share, then copy the desired configuration file to the appropriate directory on the other user's machine.

Configuration files for C++Test projects are saved at
<C++Test_install_dir>/C++TestProj/<username>/Projects.

Printing Results

You can print your results by performing the following step:

- Choose **File> Print Report**.

C++Test will then print your results.

Previewing the Printed Page

If you would like to see what the page will look like before you print it, change page properties, or change font size, chose **File> Print Preview**. This will open a dialog box which offers a print preview and lets you modify print options such as font size, paper size, page orientation, margins, and detail level. To print the page from this preview screen, click **Print**. To close the preview page without printing, click **Close**.

GUI Overview

C++Test's GUI contains the following components:

- **Menu Bar:** Contains commands that open and manage files and tests.
- **Tool Bar:** Provides access to the most common commands used to open and manage files and tests.
- **Shortcut Menus:** Provide a quick way to perform an action related to the area or item containing the shortcut menu. Options depend on the location of the shortcut menu. Most shortcut menus let you move between tabs and perform additional actions.
- **File List:** Lets you select which of the opened files you would like to work with.
- **Symbol Tree:** Lets you see a file's structure and select which symbol you want to work with.
- **Source Code Tab:** Displays the current file's source code. If a method is selected in the Symbol tree, that method will be highlighted.
- **Test Progress Tab:** Lets you record and play tests. If you are testing a single method, lets you attach a debugger.
- **Results Tab:** Displays test results. Works with Test Result Viewer (displays detail about a specific test case).
- **Test Case Editor:** Lets you create, view, and edit test cases. Works with Test Case Editor window (provides the skeleton in which you can add test case values).
- **Stub Tables Tab:** Lets you view and modify stub types, enter your own stubs, and create stub tables for C++Test generated stubs. Uses Test Case Editor window.
- **Suppressions Tab:** Lets you determine which classes and methods are tested.
- **Output Tab:** Displays all C++ Test output, including build information and testing information.

- **Messages Tab:** Guides you through the messages displayed in the Source Code tab and Results tab. It also lets you open detailed test case results in the Test Result Viewer window and the source file in the editor of your choice.
- **Customize Panel:** Lets you determine general C++Test settings such as editor options, suppression options, and source viewer properties.
- **Project Settings Panel:** Lets you determine project-wide settings such as build options, CodeWizard and Insure++ options, and ignore paths.
- **Current Settings Panels:** Lets you determine automatic test settings that are applied when testing a file, class, or function.

The Menu Bar

C++Test's menu bar contains commands that open and manage files and tests. Available commands are:

File Menu

- **Open File(s):** Opens a file chooser from which you can select one or more files. Opened files will be displayed in the File List.
- **Open DevStudio Project:** Opens a file chooser from which you can select a DevStudio project. The project's files will be displayed in the File List.
- **Close File(s):** Closes all files currently in the File List.
- **Edit Source:** Lets you edit the file currently selected in the File List. When you select this command, the file will be opened in the editor you specified in the Global Settings panel.
- **Read Symbols:** Prompts C++Test to parse the selected file, then represent its symbols in the Symbol tree.
- **Print Report:** Prints the results that are currently open in your Results tab.
- **Print Preview:** Lets you preview and modify how your results will be printed.
- **Recent Files:** Lets you open a recent file. Contains the six most recent files.
- **Recent DevStudio Projects:** Lets you open a recent DevStudio Project. Contains the six most recent projects.
- **Exit:** Closes C++Test.

Project Menu

- **New Project:** Lets you create a new C++Test project.
- **Open Project:** Opens a file chooser that lets you open a saved C++Test project.
- **Close Project:** Closes the current C++Test project.

- **Save Project:** Saves changes to the current C++Test project.
- **Save Project As:** Saves changes to the current C++Test project into a user-defined file.
- **Delete Project:** Lets you delete the current C++Test project.

Tests Menu

- **Build Test:** Tells C++Test to parse the selected file, read its symbols, instrument it (if necessary), then compile and link it.
- **Test File(s):** Opens a file chooser that lets you open one or more file(s), then automatically builds a test executable and performs white-box testing on the selected file(s).
- **Test DevStudio Project:** Opens a file chooser that lets you open a DevStudio Project, then automatically builds a test executable and performs white-box testing on the project's files.
- **Test All:** Builds an executable for all files currently in the File List, automatically generates test cases for all of these files, then executes all automatically generated test cases.
- **Enable Coverage:** Tells C++Test to generate coverage information on all tests.
- **Show Coverage:** Launches a window that displays coverage information.
- **Clear Coverage:** Removes current coverage information from the Coverage window.

Options Menu





- **Customize:** Opens the Customize panel. This panel lets you customize general C++Test operations such as editor options, suppression options, and source viewer options.
- **Project Settings:** Opens the Project Test Settings panel. This panel lets you customize project-specific C++Test operations such as build options, CodeWizard and Insure++ options, and ignore paths.
- **Test Settings:** Opens the Current Test Settings panel. This panel lets you customize how C++Test automatically creates test cases. These settings can be configured at the file, class, or function level.
- **Technical Support:** Opens the Technical Support panel. This panel lets you determine if and how C++Test automatically creates a zip file when a build fails. See “If Your Build Fails” on page 97 for more information.


Help Menu

- **Online Help:** Opens the online User’s Guide.
- **View Password:** Opens a dialog box that displays your password and expiration date.
- **About:** Opens a dialog box that displays your C++Test version number and ParaSoft contact information.

The Tool Bar

The following commands are available in the C++Test tool bar.

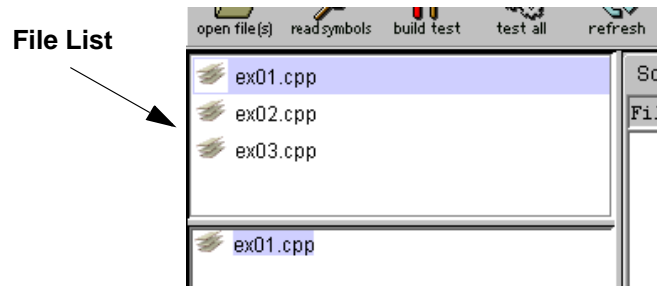
Button	Name	Action
	Open File(s)	Opens a file chooser from which you can select one or more files. Opened files will be displayed in the File List.
	Read Symbols	Parses the selected file and represents its symbols in the Symbol tree. After you have performed a test, clicking this button restores test results.
	Build Test	Parses the selected file, reads its symbols, instruments it if necessary, then compiles and links it.
	Test All	Builds an executable for all files currently in the File List, automatically generates test cases for all of these files, then executes all automatically generated test cases.

	Refresh	Recreates your Symbol tree from its current symbol repository, closes all expanded nodes, and deselects the currently selected node. If the file is closed, it will clear the Symbol tree and other tabs.
-----------------------------------------------------------------------------------	----------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The File List

The File List contains all opened files. You can use this list to indicate which file you want to work with or apply a certain action (such as **Read Symbols** or **Build Test**) to.

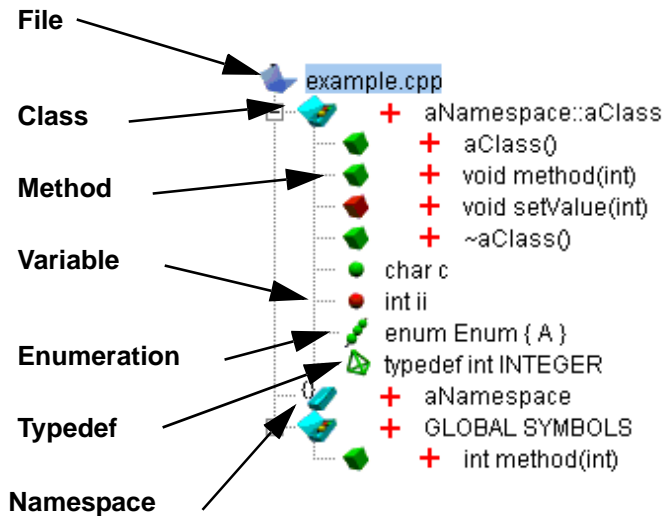
If you right-click a file in the File List, C++Test will open a shortcut menu that allows you to edit the file's source, read the file's symbols, build a test, or close the file.



The Symbol Tree

The Symbol tree represents all of the symbols in your file. You use the Symbol tree to control which file, class, or method you want to perform an action on.

The Symbol tree represents symbols in the following manner:



Public symbols are green.

Yellow symbols are protected

Red symbols are private

Gray symbols are functions that cannot be tested (for example, functions that are only declared, functions that are virtual, etc.).

If an include file is defined in a file, it will be included in the Symbol tree.

Refreshing Your Symbol Tree

To restore your Symbol tree to its original (unexpanded and deselected) state, click the **Refresh** tool bar button. C++Test will then recreate your Symbol tree from its current symbol repository, close all expanded nodes, and deselect the currently selected node. If the file is closed, this button will clear the Symbol tree and other tabs.

If you have made changes to your source file and would like your Symbol tree to reflect these changes, click the **Read Symbols** tool bar button. C++Test will then parse and instrument your files, then recreate your Symbol tree.

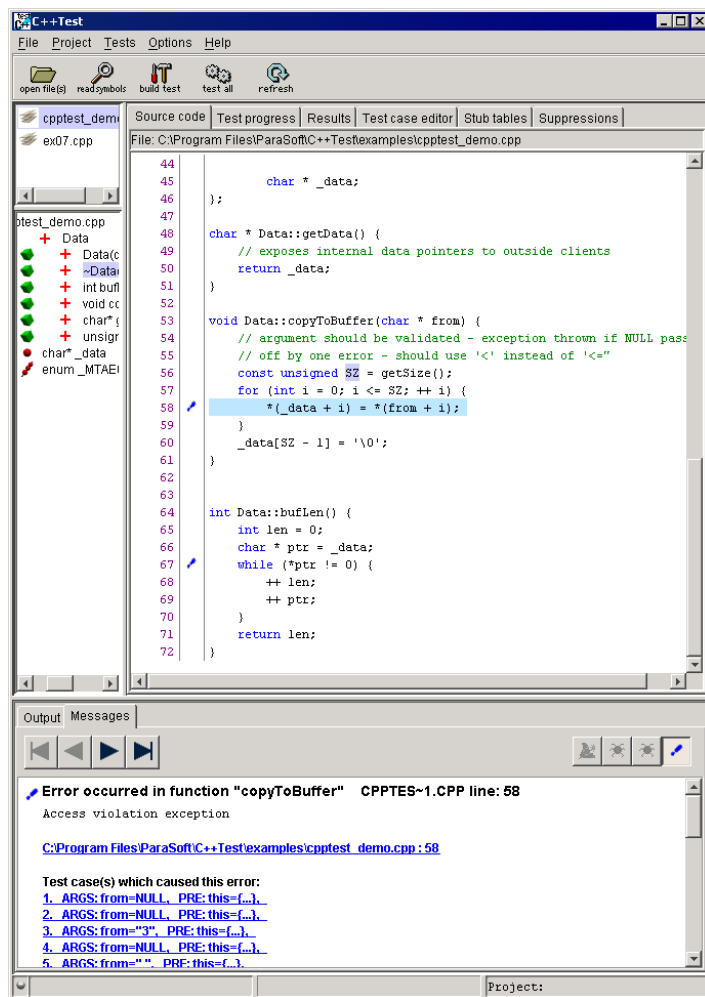
The Source Code Tab

The Source Code tab lets you view the source code of any file currently in the File List.

The Source Code tab also works with the Messages tab to correlate messages about errors found with specific lines of code.

For information on viewing your source, see “Viewing Source Code” on page 43

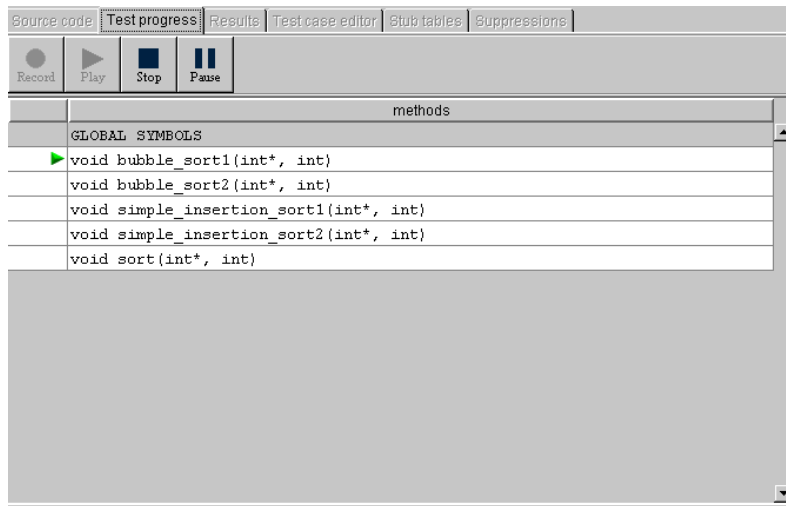
For information on viewing results in the Source Code tab, see “Viewing Details About Errors Found” on page 35.







The Test Progress Tab

The Test Progress tab allows you to record, play, stop, or pause any C++Test test. It indicates what method is currently being tested by placing a green arrow to the left of that method.

For information on running tests, see “Running a Test” on page 21.



The following commands are available in the Test Progress tab:

Button	Name	Action
	Record	Automatically generates test cases for the selected file, class, or method, then executes the automatically generated test cases.
	Play	Executes all test cases that are currently available in the Test Case Editor. Does <i>not</i> automatically generate any test cases.
	Stop	Stops the current test.
	Pause	Pauses or resumes the current test.

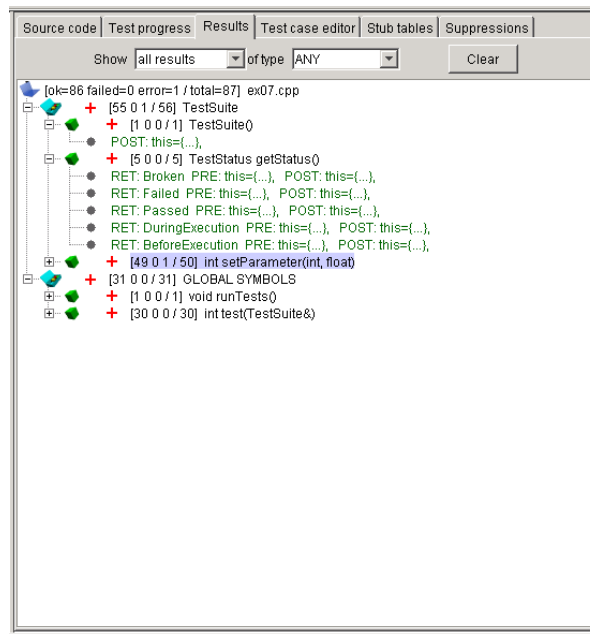
The Results Tab

The Results tab lets you view test cases and results. For information about viewing test case results, see “Test Results and Output” on page 31.

The Results tab contains a tree that represents each file, class, function, and test case.

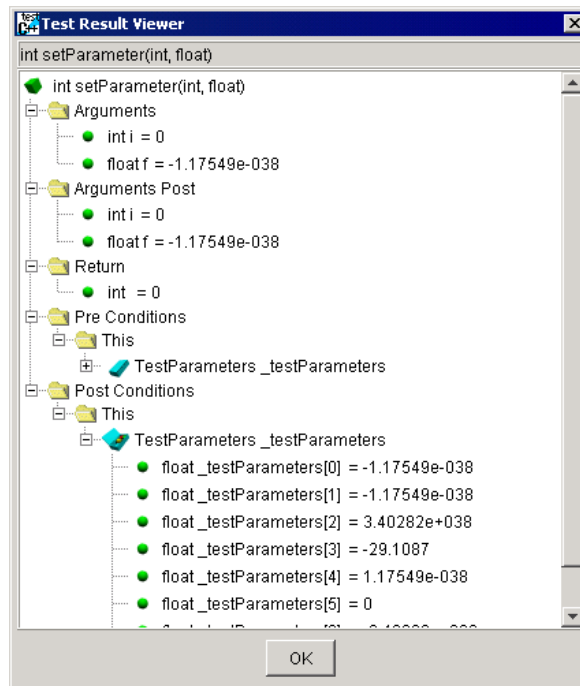
If you select a red test case node for a test case that resulted in an error, information about that error will be displayed in the Messages tab.

If you right-click any test case node in the Results tab then choose **View Test Results**, C++Test will open that test case in the Test Result Viewer window.



The Test Result Viewer Window

The Test Result Viewer window displays detailed information about a specific test case. This window can be opened by right-clicking any test case node in the Results tab then choosing **View Test Results**, or by double-clicking a link to a test case in the Messages tab.



The Test Result Viewer window might contain the following main branches (the number of branches displayed depends on the function under test):

- **Arguments:** Values in this branch represent the function's arguments.

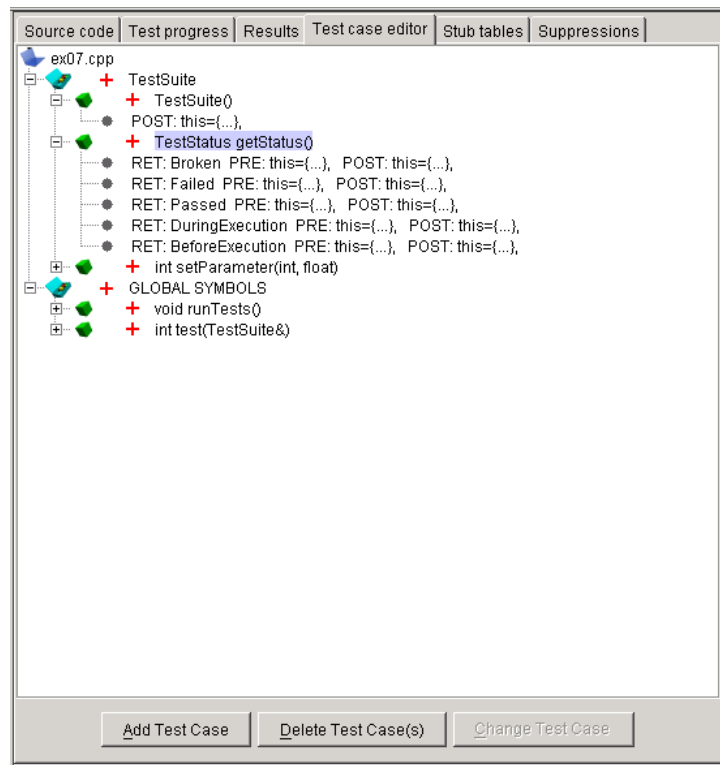
- **Arguments Post:** Values in this branch represent the function's arguments after the test.
- **Return:** Values in this branch represent the function's result.
- **Pre Conditions> This:** Values in this branch represent the condition of the object before the test.
- **Pre Conditions> Externals:** Values in this branch represent the condition of the global variables before the test.
- **Post Conditions> This:** Values in this branch represent the condition of the object after the test.
- **Post Conditions> Externals:** Values in this branch represent the condition of the global variables after the test.

The Test Case Editor Tab

The Test Case Editor tab lets you create, view, and edit test cases. For information on working with test cases, see “Entering Test Cases” on page 15.

The Test Case Editor tab contains a tree that represents each file’s functions and any test cases available for each function.

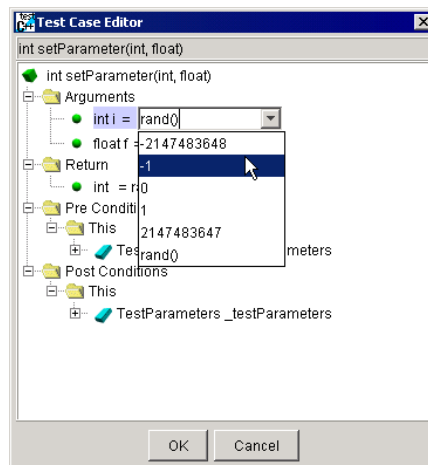
Selecting any function’s node in the Test Case Editor tab then clicking **Add Test Case** (or selecting any test case node in the Test Case Editor tab then clicking **Edit Test Case**) opens the Test Case Editor window.



The Test Case Editor Window

Test cases are added and edited in the Test Case Editor window, which can be opened in any of the following ways:

- Select a function's node in the Test Case Editor, then click the **Add Test Case** button.
- Selecting any test case node in the Test Case Editor tab, then click the **Edit Test Case** button.
- Double-click a test case link in the Messages tab.



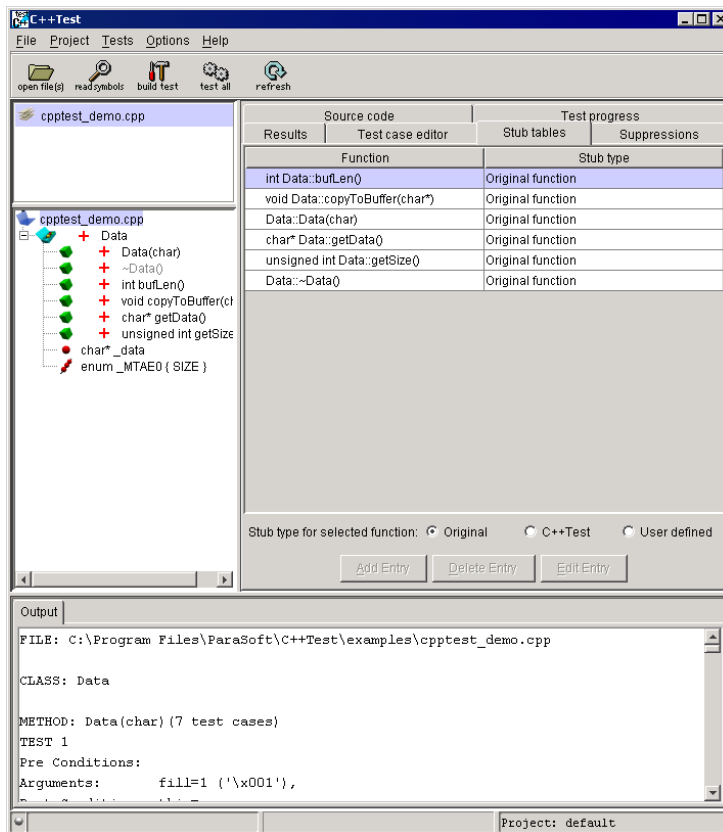
The Test Case Editor window might contain the following main branches (the number of branches displayed depends on the function under test):

- **Arguments:** Values in this branch represent the function's arguments.
- **Return:** Values in this branch represent the function's result.
- **Pre Conditions> This:** Values in this branch represent the condition of the object before the test.

- **Pre Conditions> Externals:** Values in this branch represent the condition of the global variables before the test.
- **Post Conditions> This:** Values in this branch represent the condition of the object after the test.
- **Post Conditions> Externals:** Values in this branch represent the condition of the global variables after the test.

The Stub Tables Tab

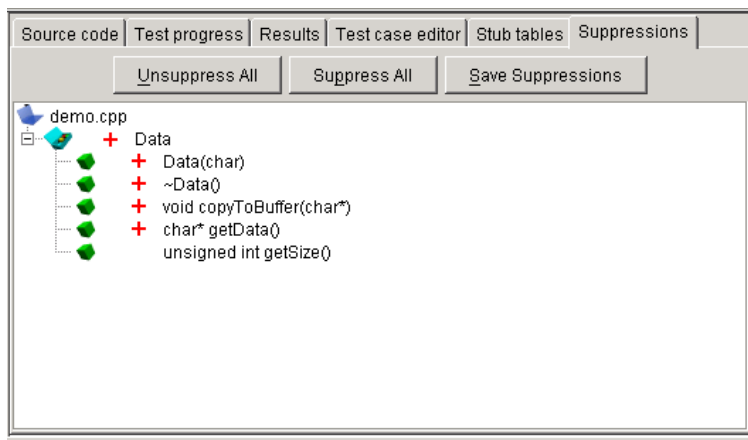
The Stub Tables tab lets you determine what type of stub is used, enter user-defined stubs, and create stub tables. For information on stubs, see “Working With Stubs” on page 47.



The Suppressions Tab

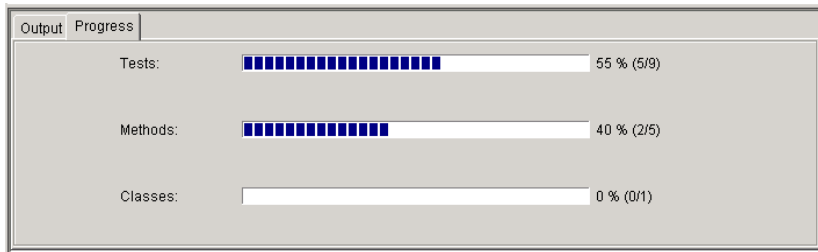
The Suppressions tab lets you control which classes and methods are tested.

For information on controlling what classes and methods are included and excluded, see “Excluding Specific Classes or Methods From Your Test” on page 51.



The Progress Tab

While a test is being performed, the Progress tab (located at the bottom of the GUI) displays information about test progress.



For information on monitoring progress, see “Monitoring Test Progress” on page 59.

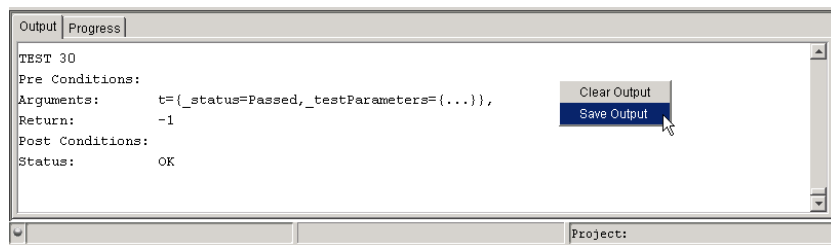
The Output Tab

This tab displays all C++Test build and test output. It is available at the bottom of the GUI during or after a build or test.

To save this tab's content as a text file, right-click any area of this tab, choose **Save Output** from the shortcut menu, then specify where you want to save the output.

To clear this tab's output, right-click any area of this tab, then choose **Clear Output** from the shortcut menu.

For information about the output contained in this tab, see "Test Results and Output" on page 31.

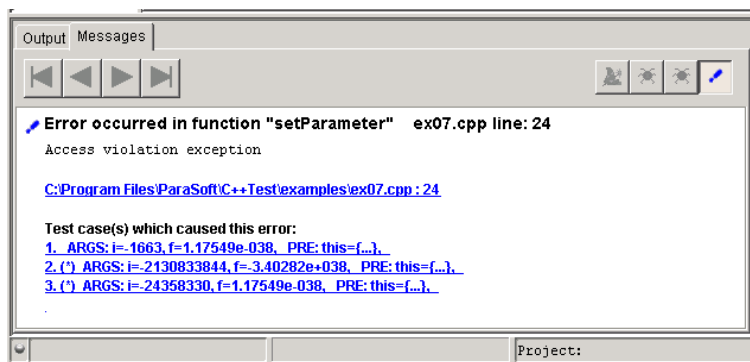


The Messages Tab

The Messages tab guides you through the messages displayed in the Source Code tab and Results tab. It also lets you open detailed test case results in the Test Result Viewer window and the source file in the editor of your choice.

The Messages tab is available in the bottom of the C++Test GUI when errors have been found and the Source Code tab is selected or an error node in the Results tab is selected.

For information about using this tab to view messages, see “Viewing Details About Errors Found” on page 35.



The Settings Panels

C++Test has three settings panels that let you customize C++Test.

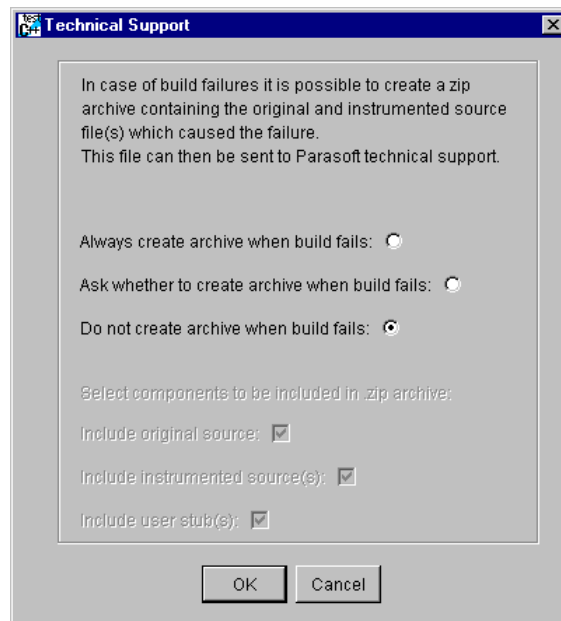
- **Customize Panel:** Lets you determine general C++Test settings such as editor options, suppression options, and source viewer properties.
- **Project Settings Panel:** Lets you determine project-wide settings such as build options, CodeWizard and Insure++ options, and ignore paths.
- **Test Configuration Panels:** Let you determine automatic test settings that are applied when testing a file, class, or function.

If Your Build Fails

If your build fails, the best way to remedy the problem is to create a zip archive containing the original and instrumented source file(s) that caused that failure, then send that zip file to ParaSoft's Quality Consultants.

To make this process as easy as possible for you, C++Test can automatically create an archive when a build fails. On average, these archives are about half a megabyte, and are created in about one minute.

By default, C++Test does not create an archive when your build fails. To modify archive creation options, open the Technical Support panel by choosing **Options> Technical Support**, then modify the archiving options.



Available options include:

- **Always create archive when build fails:** Every time that a build fails, C++Test will automatically create an archive.
- **Ask whether to create archive when build fails:** Every time that a build fails, C++Test will ask you whether or not you want it to create an archive.
- **Do not create archive when build fails:** Every time that a build fails, C++Test will not automatically create an archive or give you the option to create an archive.

If you choose to always create an archive or to be asked about archive creation, you can also set the following options that determine what the zip file contains:

- **Include original source:** Always includes the original source.
- **Include instrumented source(s):** Always includes the instrumented source(s).
- **Include user stub(s):** Always includes user-defined stub(s).

The more information the zip file includes, the better our Quality Consultants will be able to pinpoint the problem.

If a zip file is created, it will be named <testname>.zip, and it will be saved in <C++Test_installation_dir>/C++TestProj/<username>/<testname>. (The exact name and location will be displayed in the C++Test Output tab).

Send this zip file to ParaSoft's Quality Consultants at cpptest-support@parasoft.com.

Note: If you want to view the contents of the zip archive, you must have WinZip 7.0.

Contacting ParaSoft

ParaSoft is committed to providing you with the best possible product support for C++Test. If you have any trouble installing or using C++Test, please follow the procedure below in contacting our Quality Consulting department.

- Check the manual.
- Be prepared to recreate your problem.
- Know your C++Test version. (You can find it in **Help> About**).
- If the problem is not urgent, report it by e-mail or by fax.
- If you call, please use a phone near your computer. The Quality Consultant may need you to try things while you are on the phone.

C++Test experts are available online to answer your questions. To receive live online support, go to <http://www.parasoft.com/experts>.

Contact Information

- **USA Headquarters**
Tel: (888) 305-0041
Fax: (626) 305-9048
Email: cpptest-support@parasoft.com
Web Site: <http://www.parasoft.com>
- **ParaSoft France**
Tel: +33 (0) 1 64 89 26 00
Fax: +33 (0) 1 64 89 26 10
Email: quality@parasoft-fr.com
- **ParaSoft Germany**

Tel: +49 (0) 78 05 95 69 60

Fax: +49 (0) 78 05 95 69 19

Email: quality@parasoft-de.com

- **ParaSoft UK**

Tel: +44 171 288 66 00

Fax: +44 171 288 66 02

Email: quality@parasoft-uk.com

C++Test Tutorials

C++Test's tutorials offer step-by-step guides on such topics as:

- Performing white-box testing
- Performing black-box testing
- Viewing coverage information
- Working with stubs
- Performing regression testing
- Testing a Developer Studio project
- Testing templates
- Initializing objects

These tutorials are available online at

<http://www.parasoft.com/products/ctest/manuals/tutorials/index.htm>

Index

A

automatic test settings 64

B

black-box testing 22
build failure 97
building a test executable 13
buttons 76

C

C++Test
 customizing 51
 installing 2
char types 19
class test 101
 performing 101
class, testing 26
CodeWizard 13, 53
coding standards 53
compiler options 62
constructors 17
contacting ParaSoft 99
coverage 56
Current Settings panel 64
Customize panel 60
customizing C++Test 60, 62, 64, 96

D

DevStudio 10, 27, 42

E

editor, configuring 42
error prevention 53

F

failed builds 97
file list 78
files
 opening 10
 testing 27

G

GUI 71

I

ignore paths 63
include files 23, 62
installation 2
Insure++ 54

L

LicenseServer 4
licensing 2

M

menus 73
Messages tab 35, 95
methods
 single stepping 25
 testing 24

O

objects, initializing 14, 17

- opening C++Test projects 11, 67
- opening files, DevStudio projects 10
- outcomes, validating 41
- Output tab 38, 94, 95

P

- parameters, sharing 69
- ParaSoft, contacting 99
- Play 22
- pointers 17
- preprocessor flags 62
- preventing errors 53
- printing results 70
- progress 59
- Progress tab 93
- Project Settings panel 62
- projects
 - C++Test 11, 60, 67
 - DevStudio 10, 27

Q

- Quality Consulting 99

R

- read symbols 13, 73
- Record 22
- refreshing symbol tree 45
- regression testing 22, 46
- results 31, 70
- Results tab 31, 85
- runtime error detection 54

S

- Settings panels 60, 62, 64, 96
- settings, customizing 60, 62
- single stepping 25
- source
 - editing 42

- viewing 43
- Source Code tab 81
- Source Viewer 43
- string types 19
- Stub Tables tab 91
- stubs
 - about 47
 - stub tables 49
 - stub types 47
 - user-defined 48
- suppressions 51, 60
- Suppressions tab 92
- symbol tree 79
 - refreshing 45
- symbols, reading 13, 73

T

- Technical Support 99
- templates 23
- Test All 28
- Test Case Editor Tab 88
- Test Case Editor window. 88
- test cases
 - adding 15
 - copying and modifying 20
 - editing 19
 - playing 30
 - removing 20
- Test Configuration panel 64
- test executable, building 13
- Test Progress tab 83
- testing
 - black-box 22
 - classes 26
 - customizing 51, 60, 62, 64
 - DevStudio projects 27
 - files 27
 - methods 24
 - overview 7
 - regression 22, 46
 - results 31
 - running a test 21
 - template functions 23
 - validating outcomes 41

- white-box 22
- troubleshooting 97
- tutorials 101

V

- validating outcomes 41

W

- white-box testing 22