

# USER'S GUIDE

## TCAT C/C++

Version 9.2

Test Coverage Analysis  
Tool For C and C++



SOFTWARE RESEARCH, INC.

**This document property of:**

Name: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

Phone \_\_\_\_\_



625 Third Street  
San Francisco, CA 94107-1997  
Tel: (415) 957-1441  
Toll Free: (800) 942-SOFT  
Fax: (415) 957-0730  
E-mail: support@soft.com  
<http://www.soft.com>

ALL RIGHTS RESERVED. No part of this document may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise without prior written consent of Software Research, Inc. While every precaution has been taken in the preparation of this document, Software Research, Inc. assumes no responsibility for errors or omissions. This publication and features described herein are subject to change without notice.

**TOOL TRADEMARKS:** CAPBAK/MSW, CAPBAK/UNIX, CAPBAK/X, CBDIFF, EXDIFF, SMARTS, SMARTS/MSW, S-TCAT, STW/Advisor, STW/Coverage, STW/Coverage for Windows, STW/Regression, STW/Regression for Windows, STW/Web, TCAT, TCAT C/C++ for Windows, TCAT-PATH, TCAT for JAVA, TCAT for JAVA/Windows, TDGEN, TestWorks, T-SCOPE, Xdemo, Xflight, and Xvirtual are trademarks or registered trademarks of Software Research, Inc. Other trademarks are owned by their respective companies. METRIC is a trademark of SET Laboratories, Inc. and Software Research, Inc. and STATIC is a trademark of Software Research, Inc. and Gimpel Software.

Copyright © 1998 by Software Research, Inc  
(Last Update December 18, 1998)

/home/l1/wu/unix-tcat/tcatwu98/tcat9.book

# Table of Contents

---

<b>Preface</b> .....	<b>xv</b>
<b>CHAPTER 1 Overview of TCAT C/C++</b> .....	<b>1</b>
1.1 Basics of Automated Software Test Methods .....	1
1.2 Coverage Analysis Tools .....	2
1.2.1 Technical Summary .....	2
1.2.2 Technology .....	3
1.2.3 Functionality .....	4
1.2.4 Effectiveness Assessment .....	7
1.3 TCAT C/C++ .....	8
1.3.1 Feature Summary .....	9
Instrumentor .....	9
Runtime .....	9
Coverage .....	9
1.3.2 File Inventory .....	10
1.3.3 Installation Process .....	11
<b>CHAPTER 2 Quick Start</b> .....	<b>13</b>
2.1 Overview .....	13
2.2 STEP 1: Starting Up TCAT C/C++ .....	14
2.3 STEP 2: Selecting Load Setting .....	17
2.4 STEP 3: Instrumenting the Example Application .....	18
2.5 STEP 4: Choosing and Linking a Runtime Version .....	20
2.6 STEP 5: Running the Application .....	22
2.7 STEP 6: Opening the Coverage Window .....	24
2.8 STEP 6: Choosing a Trace File .....	25
2.9 STEP 8: Generating and Viewing a Report .....	26

---

---

**TABLE OF CONTENTS**

---

<b>2.10</b>	<b>STEP 9: Viewing a Logical Branch's Source Code .....</b>	<b>28</b>
<b>2.11</b>	<b>STEP 10: Sign Off and Cleanup .....</b>	<b>29</b>
<b>2.12</b>	<b>Summary .....</b>	<b>30</b>

**CHAPTER 3 TCAT C/C++****Graphical User Interface. . . . . 31**

<b>3.1</b>	<b>TCAT C/C++ .....</b>	<b>31</b>
<b>3.2</b>	<b>Invoking TCAT C/C++ .....</b>	<b>31</b>
<b>3.3</b>	<b>TCAT C/C++ Main Window .....</b>	<b>32</b>
3.3.1	File.....	33
3.3.2	Options .....	33
3.3.3	File Selection Section.....	33
3.3.4	Utilities Section .....	33
3.3.5	Shell Section .....	33
<b>3.4</b>	<b>File Pull-Down Menu .....</b>	<b>34</b>
3.4.1	Load Setting .....	35
3.4.2	Save Setting .....	35
3.4.3	Set Project DB .....	35
3.4.4	Exit .....	35
<b>3.5</b>	<b>Options Pull-Down Menu .....</b>	<b>36</b>
3.5.1	Instrumentor/Compiler Options.....	36
	Instrumentor Options . . . . .	37
	Compiler Options . . . . .	37
<b>3.6</b>	<b>Link/Build/Run Options .....</b>	<b>38</b>
3.6.1	Link Options .....	39
	Command . . . . .	39
	Flags . . . . .	39
	Library Flags . . . . .	39
	Runtime Module . . . . .	39
3.6.2	Build Options .....	40
	Command . . . . .	40
	Flags . . . . .	40
3.6.3	Run Options .....	41
	Target Name . . . . .	41
	Command Line Args . . . . .	41
<b>3.7</b>	<b>File Selection Section .....</b>	<b>42</b>
3.7.1	Directories .....	43
3.7.2	Files.....	43
3.7.3	Selection .....	43
3.7.4	Clear All .....	43
3.7.5	Select All.....	43

---

## TABLE OF CONTENTS

---

<b>3.8</b>	<b>Utilities Section</b>	<b>44</b>
3.8.1	Instrument	45
3.8.2	Link	45
3.8.3	Build	45
3.8.4	Run	45
3.8.5	Link & Run	45
3.8.6	Build & Run	45
3.8.7	Xcover	45
3.8.8	Compile	46
3.8.9	Xdigraph	46
3.8.10	Xcalltree	46
3.8.11	Vi	46
3.8.12	Rm	46
3.8.13	Debug	46
3.8.14	Lint	47
3.8.15	SCCS get	47
3.8.16	SCCS edit	47
3.8.17	SCCS delta	47
<b>3.9</b>	<b>Shell Section</b>	<b>48</b>
3.9.1	Command	48
3.9.2	Message Area	48
3.9.3	Kill	48
 <b>CHAPTER 4 Runtime System</b>		 <b>49</b>
4.1	TCAT C/C++ “Runtime” Support Options	49
4.2	Pre-Selected Link-Time Options	50
4.3	Performance Gain With Buffering	51
 <b>CHAPTER 5 Xcover — Coverage Analyzer</b>		 <b>53</b>
5.1	Xcover	53
5.2	Xcover Functionality	53
5.3	Command Line Invocation	54
5.4	Xcover Sample Screens	55
5.5	Operation of Xcover	57
5.5.1	Xcover Options	58
5.5.2	Selecting a Trace File	58
5.5.3	Selecting an Archive File	58

---

**TABLE OF CONTENTS**

---

<b>CHAPTER 6 Xcalltree Utility</b>	<b>59</b>
6.1 Purpose	59
6.2 Xcalltree File Format	59
6.3 Invoking Xcalltree	60
6.4 Xcalltree Main Window	61
6.4.1 File	62
6.4.2 Options	62
6.4.3 Zoom In & Zoom Out	62
6.4.4 View Source	62
6.4.5 Statistics	62
6.4.6 Print	62
6.4.7 Annotation	63
6.4.8 Help	63
6.5 File Pull-Down Menu	64
6.5.1 Load New Graph	65
6.5.2 Load New Multi Graph	65
6.5.3 Set Archive	65
6.6 Calltree File Selection Dialog Box	66
6.6.1 Filter	67
6.6.2 Directories	67
6.6.3 Files	67
6.6.4 Selection	67
6.6.5 OK	67
6.6.6 Filter Button	67
6.6.7 Cancel	67
6.7 Option Window	68
6.7.1 Zoom Scale	69
6.7.2 Horizontal Spacing	69
6.7.3 Depth	69
6.7.4 Root Name	70
6.7.5 Edge Characteristics	72
Edge Color	72
Unhighlighted Edge	72
Display Mode	72
6.7.6 Node Characteristics	73
Size	73
Aspect Ratio	73
Default Color	73
Low-level Color	73
Normal Color	73
High-level Color	73
Apply	73
6.8 Zoom In & Zoom Out Options	74

---

**TABLE OF CONTENTS**

---

6.9	View Source Window .....	75
6.9.1	Description of Source Code Viewing .....	76
6.10	Statistics Window .....	77
6.10.1	Links.....	78
6.10.2	Call pairs.....	78
6.10.3	Modules/Depth .....	78
6.10.4	Recursive.....	78
6.11	Print Window .....	79
6.11.1	Paper Size Information .....	80
6.11.2	Enlargement Factors .....	81
6.11.3	Font Information .....	82
6.11.4	Print Locator.....	82
6.12	Annotation Window .....	83
6.12.1	Threshold 1 & Threshold 2.....	84
6.12.2	None .....	84
6.12.3	S0.....	84
6.12.4	Ninvokes .....	84
6.12.5	S1.....	84
6.12.6	C1 .....	84
6.12.7	Cyclo .....	84
6.12.8	Nsegs .....	84
6.12.9	Npairs.....	85
6.12.10	Nlines .....	85
6.12.11	Npaths.....	85
6.12.12	User .....	85
6.12.13	Connections .....	85
6.12.14	Apply .....	85
6.12.15	Reset .....	85
6.12.16	Close .....	86
6.12.17	Help .....	86
6.13	Quick Reference Guide to Xcalltree Annotations .....	87
 <b>CHAPTER 7 Xdigraph Utility . . . . .</b>		<b>89</b>
7.1	Purpose .....	89
7.2	Xdigraph File Format .....	89
7.3	Invoking Xdigraph .....	90
7.4	Xdigraph Main Window .....	92
7.4.1	File.....	93
7.4.2	Options .....	93
7.4.3	Zoom In .....	93
7.4.4	Zoom Out.....	93
7.4.5	View Source.....	93
7.4.6	Statistics .....	93

---

**TABLE OF CONTENTS**

---

7.4.7	Print.....	93
7.4.8	Annotation .....	94
7.4.9	Help .....	94
7.5	<b>File Pull-Down Menu .....</b>	<b>95</b>
7.5.1	Load New Graph .....	96
7.5.2	Load New Module .....	96
7.5.3	Set Archive .....	96
7.5.4	Exit .....	96
7.5.5	<b>Digraph File Message Box .....</b>	<b>97</b>
	Filter . . . . .	98
	Directories . . . . .	98
	Files . . . . .	98
	Selection . . . . .	98
	OK . . . . .	98
	Filter Button . . . . .	98
	Cancel . . . . .	98
7.6	<b>Options Window .....</b>	<b>99</b>
7.6.1	<b>Zoom Scale.....</b>	<b>100</b>
7.6.2	<b>Node Characteristics .....</b>	<b>101</b>
	Shape . . . . .	101
	Size . . . . .	101
	Vertical Spacing . . . . .	101
	Aspect ratio . . . . .	101
7.6.3	<b>Edge Characteristics .....</b>	<b>102</b>
	Unhighlighted Edge . . . . .	102
	Eccentricity . . . . .	102
	Default Color . . . . .	102
	Low-level Color . . . . .	102
	Normal Color . . . . .	102
	High-level Color . . . . .	102
	Apply . . . . .	103
	Reset . . . . .	103
	Close . . . . .	103
	Help . . . . .	103
7.7	<b>Zoom In/Zoom Out Window .....</b>	<b>104</b>
7.8	<b>View Source Window .....</b>	<b>105</b>
7.9	<b>Statistics Window .....</b>	<b>106</b>
7.9.1	<b>File Name.....</b>	<b>107</b>
7.9.2	<b>Node and Edge Count .....</b>	<b>107</b>
7.9.3	<b>Cyclomatic Number (Cyclomatic Complexity) .....</b>	<b>107</b>
7.9.4	<b>Average, Minimum and Maximum Path Lengths.....</b>	<b>107</b>
7.9.5	<b>Path Count by Iteration Groups.....</b>	<b>107</b>



---

**TABLE OF CONTENTS**

---

<b>7.10</b>	<b>Print Window .....</b>	<b>108</b>
7.10.1	Paper Size Information .....	109
7.10.2	Enlargement Factors .....	110
7.10.3	Font Information .....	111
7.10.4	Print Locator.....	111
<b>7.11</b>	<b>Annotation Window .....</b>	<b>112</b>
7.11.1	Threshold 1 & 2 .....	113
7.11.2	None .....	113
7.11.3	Nhits .....	113
7.11.4	N% .....	113
7.11.5	Nlines .....	113
7.11.6	User .....	113
7.11.7	Highlight .....	114
7.11.8	Path File .....	114
7.11.9	Apply .....	114
7.11.10	Reset .....	114
7.11.11	Close .....	114
7.11.12	Help .....	114
7.11.13	Colors.....	115
<b>7.12</b>	<b>Quick Reference Guide to Xdigraph Annotations .....</b>	<b>116</b>
 <b>APPENDIX A C/C++ Instrumentor Engine . . . . .</b>		<b>117</b>
 <b>APPENDIX B Resource File Variables . . . . .</b>		<b>131</b>
 <b>APPENDIX C cover —TCAT C/C++’s Coverage Analyzer . . . . .</b>		<b>147</b>

---

## *TABLE OF CONTENTS*

---

# List of Figures

---

FIGURE 1	TCAT C/C++ Graphical Displays . . . . .	5
FIGURE 2	Setting Up the Display (Initial Condition) . . . . .	14
FIGURE 3	The TCAT C/C++ Main Window . . . . .	16
FIGURE 4	Selecting the Load Setting . . . . .	17
FIGURE 5	Instrumenting the Source File . . . . .	19
FIGURE 6	Selecting the Runtime Object Module . . . . .	21
FIGURE 7	Running Motifburger . . . . .	23
FIGURE 8	The Coverage Window . . . . .	24
FIGURE 9	Selecting a Trace File Name . . . . .	25
FIGURE 10	Coverage Information for Each Function . . . . .	26
FIGURE 11	Looking at Source Code . . . . .	28
FIGURE 12	Completing a TCAT C/C++ Session. . . . .	29
FIGURE 13	TCAT C/C++ main window. . . . .	32
FIGURE 14	File pull-down menu . . . . .	34
FIGURE 15	Instrumentor/Compiler Options window . . . . .	36
FIGURE 16	Link/Build/Run Options window. . . . .	38
FIGURE 17	File Selection section of TCAT C/C++ main window. . . . .	42
FIGURE 18	Utilities section of TCAT C/C++ main window. . . . .	44
FIGURE 19	Shell section of the TCAT C/C++ window . . . . .	48
FIGURE 20	Xcover Example Output 1 . . . . .	55
FIGURE 21	Xcover Example Output 2 . . . . .	56
FIGURE 22	Xcalltree main window . . . . .	61
FIGURE 23	File Pull-Down Menu . . . . .	64
FIGURE 24	Calltree File Selection Dialog Box . . . . .	66
FIGURE 25	Option window . . . . .	68
FIGURE 26	Root Name Selection window example 1 . . . . .	70

---

---

## LIST OF FIGURES

---

FIGURE 27	Root Name Selection Window Example 2 .....	71
FIGURE 28	Zoom In Option illustrated. ....	74
FIGURE 29	View Source Window .....	75
FIGURE 30	Statistics Window. ....	77
FIGURE 31	Print Window .....	79
FIGURE 32	Annotation Window .....	83
FIGURE 33	Program edges as represented in a digraph .....	91
FIGURE 34	Xdigraph main window .....	92
FIGURE 35	Digraph File Pull-Down Menu .....	95
FIGURE 36	Digraph File Message Box. ....	97
FIGURE 37	Xdigraph Options Window .....	99
FIGURE 38	Zoom In feature illustrated .....	104
FIGURE 39	View Source Option Window. ....	105
FIGURE 40	Statistics Option Window .....	106
FIGURE 41	Print Dialog Window .....	108
FIGURE 42	Annotation Thresholds Window .....	112
FIGURE 43	Sample Annotation for User Threshold .....	115

---

## *LIST OF FIGURES*

---

---

## *LIST OF FIGURES*

---

# Preface

---

## Congratulations!

By choosing the TestWorks suite of testing tools, you have taken the first step in bringing your application to the highest possible level of quality.

Software testing and quality assurance, while increasingly important in today's competitive marketplace, can dominate your resources and delay your product release. By automating the testing process, you can assure the quality of your product without needlessly depleting your resources.

Software Research, Inc. believes strongly in automated software testing. It is our goal to bring your product as close to flawlessness as possible. Our leading-edge testing techniques and coverage assurance methods are designed to give you the greatest insight into your source code.

**TCAT C/C++** is a quick and easy way to detect weaknesses in your code. Easily accessible click-and-point reports find the segments that need further testing. Digraphs and calltrees visualize the location, allowing you to make immediate improvements to the structure and performance of your software.

TestWorks is the most complete solution available, and the peace of mind it provides our customers is our most valued feature.

Thank you for choosing TestWorks.

## Audience

This manual is intended to aid software testers who are using *TCAT C/C++* tools.

## Contents of Chapters

This manual is organized to aid you after installation has been completed. (See the *Installation Instructions* if you are trying to install.)

This manual is divided into the following sections:

- |           |                                                                                                                                                                                                                         |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Chapter 1 | <i>OVERVIEW OF TCAT C/C++</i> provides an introduction to test coverage tools and TCAT Version 9 including TCAT C/C++'s new features and enhancements.                                                                  |
| Chapter 2 | <i>QUICK START</i> provides a quick start to TCAT C/C++ using a demonstration test case. This chapter applies to all editions of TCAT C/C++.                                                                            |
| Chapter 3 | <i>TCAT GRAPHICAL USER INTERFACE</i> describes the TCAT C/C++ graphical user interface(GUI).                                                                                                                            |
| Chapter 4 | <i>RUNTIME SYSTEM</i> is a guide to TCAT C/C++'s Runtime Options and applies to all editions of TCAT C/C++.                                                                                                             |
| Chapter 5 | <i>XCOVER—Coverage Analyzer</i> is a guide to TCAT's complete coverage analyzer for branch (C1) or call pair (S1) metrics with a highly flexible graphical display. This chapter applies to all editions of TCAT C/C++. |
| Chapter 6 | <i>X CALLTREE UTILITY</i> explains this utility, which is a graphical display of the relationship between two called functions. This chapter applies to all editions of TCAT C/C++.                                     |
| Chapter 7 | <i>XDIGRAPH UTILITY</i> explains TCAT's graphical utility for understanding a program's structure and flow. This chapter applies to all editions of TCAT C/C++.                                                         |



## **Typefaces**

The following typographical conventions are used in this manual.

**boldface** Introduces or emphasizes a term that refers to **STW**'s window, its sub-menus and its options.

*italics* Indicates the names of files, directories, pathnames, variables, and attributes. Italics is also used for manual, chapter, and book titles.

"Double Quotation Marks" Indicates chapter titles and sections. Words with special meanings may also be set apart with double quotation marks the first time they are used.

`courier` Indicates system output such as error messages, system hints, file output, and *CAPBAK/X*'s keysave file language.

### **Boldface Courier**

Indicates any command or data input that you are directed to type. For example, prompts and invocation commands are in this text. (**stw**, for instance, invokes *STW*.)

---

*PREFACE*

---

# Overview of TCAT C/C++

This chapter is an introduction to test coverage tools and TCAT C/C++ Version 9, including TCAT C/C++'s new features and enhancements. TCAT C/C++ combines the functionality of previous versions of TCAT and S-TCAT. This chapter applies to all editions of TCAT C/C++.

---

## 1.1 Basics of Automated Software Test Methods

Mechanical assistance in software testing has been used for many years, but only recently has the essential requirement for automation become a main enabling force. Here are the main capabilities of a typical software test automation tool kit:

- **Regression Testing.** Automatic test execution — accomplished with a variety of methods — permits rapid re-testing after program changes, replays user sessions automatically, and assesses the impact of change.
- **Test Coverage.** Test coverage ensures that sets of tests are as complete as possible, measured against a range of high-quality test metrics. The products provide feedback, in real time if needed, on where current tests are inadequate.
- **Static Analysis.** Static analysis provides insight into the source code to help allocate resources to areas of the code that are more likely to be error prone, or should be rewritten because they will be difficult to maintain. Typical features include standard metrics such as control flow complexity, data complexity, and syntactical and semantic analysis.

The above order tracks the way most organizations are introducing test automation. Almost everyone understands running a test case, so automating that process using regression is a straightforward first step, and one can quickly see the benefits in reduced time and resource requirements for testing. Coverage analysis follows next, because once a test suite has been created, it is important to ask how complete the suite is, whether it is really testing all of the code, and, if it is not, which parts still need to be tested. The detailed source code information revealed through static analysis can indicate the complexity of an application, which can be used to allocate resources during development.

## 1.2 Coverage Analysis Tools

### 1.2.1 Technical Summary

A set of tests is only effective if it achieves some measurable completeness criteria. Relying only on intuition, most testers will under-test software products by 50 to 75 percent.

Coverage analyzers attack this problem by giving a numerical value to the completeness of a set of tests. There are three levels of test completeness metric:

- **C1, or branch coverage.** Used for detailed unit testing of systems of up to several hundred functions/modules. This is the most common kind of coverage analysis. (It is stronger than the commonly used but misleading **C0** metric, which counts statements exercised.)
- **S1, or call pair coverage.** Used for system interface coverage checking – to make sure every interface is fully exercised. This is a “procedure-to-procedure” coverage metric.
- **Ct, or equivalence class coverage.** Used for true path coverage of critical software modules (usually no more than 10 to 15 percent of all modules). Path classes are measured to the nearest loop repetition count by **TCAT-PATH**.

### 1.2.2 Technology

To obtain coverage information, probes are placed into an application to record which parts of the source code/application have been executed. Inserting the probes into the code is known as instrumentation, and it can be done in either the source code or the object code.

Normally, only statement coverage (**C0**, which lines of code have been exercised) can be obtained with object code insertion. This is useful, but there are many situations where 100% statement coverage will not tell whether all of the code has been tested. Below is a simple example of this situation in “C”:

```
if (index > 0)
    index--;
```

In this example, if the variable `index` is `> 0`, then both lines will be marked as executed. There is now no way to know if you ever tried this code with `index < 0` so that the second statement was not executed. On the other hand, if you used branch coverage (**C1**), it would report coverage for both the true and false condition of this `if` statement.

Source code instrumentation allows you to examine branch coverage (**C1**), call pair coverage (**S1**) and path coverage (**Ct**). Instrumentation makes a pass through the source code prior to compilation, inserting function calls to a runtime library at each branch or call pair. The key issue with instrumentation is how completely and efficiently you can process the source code.

Most tools on the market use a LEX/YACC parser generator to create a parser to understand the source code language. These parsers work well as long as you are not trying to handle a wide variety of dialects, or some of the very complex language features of newer languages such as “C++”. In order to handle these, you need true compiler technology, such as the recursive descent compiler technology Software Research uses in its current release of STW/Coverage.

State-of-the-art compiler technology delivers a number of benefits in source code instrumentation. The instrumentation process itself is much faster, and can handle errors in the source code, providing the same level of error message reporting as a compiler would. This technology is also very flexible and powerful, so that a wide variety of language dialects can be handled. With languages like “C” and “C++”, compiler vendors have added many of their own special reserved words and constructs that are not part of any standard but still must be supported by coverage tools.

### 1.2.3 Functionality

**TCAT C/C++** measures structural test completeness at the module level using the “logical segment” or C1 metric; it also measures system interface test coverage using the function call pair, or S1, metric with source instrumentation. TCAT is available for most compilers of “C” and “C++.”

**TCAT C/C++** coverage reports can be tailored to show a variety of data, including:

- segments hit
- segments not-hit
- segments newly hit
- segments newly missed
- past-test and cumulative coverage percentages

There is runtime support for analysis of cooperating processes and for cross-compilation. In addition, reports for **S1** coverage show call pairs hit, call pairs not-hit, call pairs newly hit, call pairs newly missed, past-test and cumulative coverage percentages, linear and logarithmic histograms, and coverage on listings.

**TCAT** also has utilities that aid in analyzing the call pair structure — the “call tree” of the system being analyzed. A utility called **Xcalltree** allows the user to analyze the call tree.

**TCAT C/C++ Ver. 9** combines the functionality of previous releases of **TCAT** and **S-TCAT**. Two other SR products that are also bundled with **STW/Coverage**, **TCAT-PATH** and **T-SCOPE**, are described below.

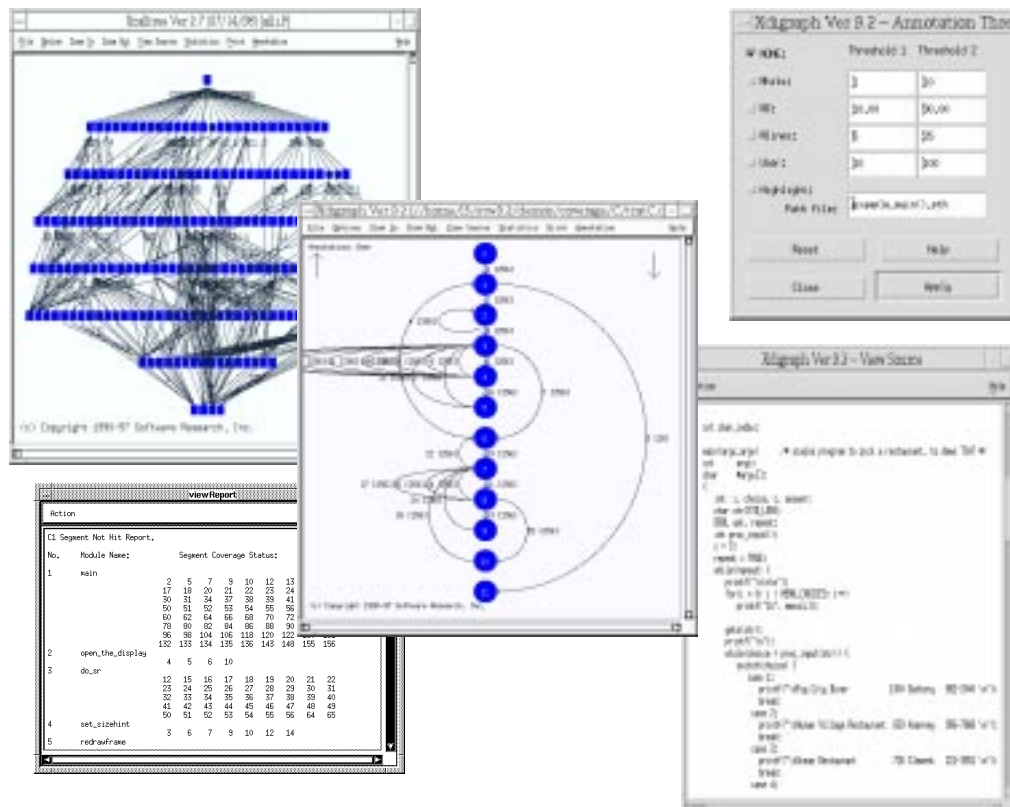
**TCAT-PATH: Path Test Coverage Analysis Tool.** **TCAT-PATH** measures module level test coverage at the path level (execution equivalence class, or Ct metric) using source instrumentation techniques. A proprietary algorithm is used to generate a set of all possible execution paths for each module. This method involves grouping “paths” — i.e., sequences of segments — into equivalence classes such that: (a) every possible program execution is included in at least one equivalence class; and (b) the number of classes is a good “compromise” size, neither trivial nor too expansive. A 250-line program in “C,” when reasonably well-coded, might produce approximately 50-100 path classes, each of which requires a separate test.

The algorithm produces classes that almost require “one test for each formal verification condition” — as if the module were being proved correct.

The coverage reports show the usual kinds of data, referring to the set of paths. The utilities that draw digraph pictures, highlight paths, and calculate the **cyclomatic complexity** are very useful in understanding the detailed structure of a module.

**T-SCOPE: Test Data Observation and Analysis System.** T-SCOPE works with any SR coverage analysis system under X Windows graphics interfaces. This system facilitates visualization of branch coverage (C1) data, call pair coverage (S1) data, and (path coverage (Ct) data. The user can program this system to present displays that include strip charts, histograms, dynamic call trees, and annotated directed graphs (flow charts).

The effects on tester productivity are impressive. Two results of the instantaneous coverage being shown in real-time are: (a) the tester can concentrate quickly on the less thoroughly-tested parts of the program, which increases test efficiency; and (b) the tester can identify which program parts relate to what is going on dynamically inside the application — an entirely new capability that can minimize test redundancy.



**FIGURE 1**

TCAT C/C++ Graphical Displays

*STW/Coverage dynamically generates a program's call-tree (top left), a module's directed graph (center), and a static report of the unexercised logical branches (bottom left) with source code displayed for an un-hit logical branch.*





#### 1.2.4 Effectiveness Assessment

Studies show that branch coverage of 85 percent or better tends to identify twice the number of defects that would have been found by “intuitive” testing. Call pair coverage of 90 percent or better will detect 25 percent more defects. Both of these techniques work because they automate and quantify a process that has relied too long on programmer and tester intuition. Automated testing focuses attention on untested parts of programs, where the latent defects lie.

The branch coverage metric is often best used when doing “unit testing,” roughly defined as working on 1,500-7,500 lines of code (1.5 to 7.5K LOC). Branch coverage tends to find single-segment faults, and the detection method is straightforward: “wrong” output most of the time.

A good practice is to use call pair coverage on the entire system, e.g. including those with over 1 million LOC to process at one time. The call pair coverage tends to identify interface errors — which are almost always self-evident when the caller-callee relationship is actually exercised. In fact, call pair coverage reduces interface defects by a factor of three to five. Call pair coverage is about 20% less difficult to obtain per KLOC than branch coverage.

For critical modules — usually a selected ten to fifteen percent of the total volume of an application — path coverage with **TCAT-PATH** can be used to extend coverage close to the practical limit.

This can be a complicated process, so it is often limited to the smaller applications, and to those with life-critical functions (such as medical products), and real-time controllers.

Completing a set of path tests can take eight to ten times as much work as branch coverage for the same volume of code, but the result is very effective. Defect detection efficiencies at 90 percent or better have been observed, even with the **Ct** coverage limited to about 75 percent. It should be noted that a 100 percent complete path-coverage test set constitutes a set of tests that match one-for-one, with the set of formal verification conditions one would use in a formal proof.

Closing the feedback loop — making it as easy as possible for programmers and/or testers to complete the test process — is a clear productivity booster. Software Research has had very good results in using both static and dynamic test visualization, in which flowcharts and call trees graphically display coverage data that is generated in real time or after the test run. This kind of display shows very quickly what is — and what is not — being exercised by a set of tests.

### 1.3 TCAT C/C++

TCAT C/C++ consists of the following:

- Tracefile formats with simpler messages for more flexibility. Ultimately these tracefiles are expected to support multi-tasking, as well as multi-threading.
- New options in selection of the **runtime** support module, with many new options and capabilities.
- Revised and updated consolidated **Xcover** utility that adapts to the new tracefile and archive file formats while preserving the prior reporting capabilities.
- A new **Xcover** X-Window-based coverage analyzer that shows coverage interactively on the screen at varying levels of detail.

### **1.3.1 Feature Summary**

The following summarizes the main enhancements to **TCAT C/C++**, over earlier versions.

#### **1.3.1.1 Instrumentor**

- C and C++ consolidated processing
- C1 + S1 consolidated coverage
- Enhanced error recovery during instrumentation
- High-speed, high-capacity system
- Instrumentor database available as a “C” structure
- “Invisible” instrumentor operation as a wrapper on **cc** or **CC**
- Inline instrumentation directives to control type and detail level of instrumentation

#### **1.3.1.2 Runtime**

- Low runtime overhead
- Support for multi-tasking
- Support for multi-threading

#### **1.3.1.3 Coverage**

- Compatible **cover** command
- New **Xcover** interactive coverage analyzer

### 1.3.2 File Inventory

Following is an inventory of the files supplied with the program:

<i>ic9/icpp9</i>	These files represent the new combined “C” and “C++” Instrumentor. They process ANSI C, K&R C, and C++ at the level of AT&T's 3.0 release. <b>ic9/icpp9</b> call the C or C++ compiler and pass through all compiler switches that are supplied by the user. Control of the instrumentation process with command line switches requires prefixing <b>-TCAT</b> to all switches specific to the instrumentation process.
<i>crunN.o</i>	Where <i>N</i> is 0-5. The supplied version of the runtime support module <i>crun5.o</i> combines C and C++ support, and combines C1 (branch) and S1 (call pair) coverage. In addition, it does all of its data collection with in-place buffering. The attached documentation gives the full description of the capabilities of the runtime system.
<i>cover</i>	This version of <b>cover</b> reads the new tracefile format.
<i>Xcover</i>	This is the new interactive GUI-based coverage analyzer.
<b>Xtcat</b>	This is the new GUI that provides access to the various utilities.
<b>Xcalltree</b>	This program draws calltrees based on information from the tracefiles and archive files produced by instrumentation.
<b>Xdigraph</b>	This program draws directed graphs based on information from the tracefiles and archive files produced by instrumentation.

### **1.3.3      Installation Process**

An installation manual is included in the “Open Me First” packet shipped with TCAT C/C++.



# Quick Start

This chapter explains getting started with **TCAT C/C++** using a demonstration test case. This chapter applies to all editions of **TCAT C/C++**.

---

## 2.1 Overview

This application note will familiarize you with the main activities involved in using **TCAT C/C++**, including instrumenting, compiling, linking and running the target program, and finally, looking at resulting coverage reports, calltree graphs and digraphs.

The program used to illustrate the operation of **TCAT C/C++** is *Motifburger*, which allows you to order various hamburger combinations. By selecting various meal combinations in an instrumented application of *Motifburger*, you exercise various logical branches or segments, creating trace files from which the coverage reports are generated.

If you are a first-time **TCAT C/C++** user, this chapter is best used if you refer to the various chapters for in-depth operational instructions. If you are an intermediate user, this chapter is best used if you refer only to those menu definitions which need further explanation.

## 2.2 STEP 1: Starting Up TCAT C/C++

Before you begin, make sure you are in the X Window System running a window manager (e.g., mwm, olwm).

Initialize an xterm-type window to serve as the **TCAT C/C++** invocation window.

During initiation of this session, the display should look like this:



---

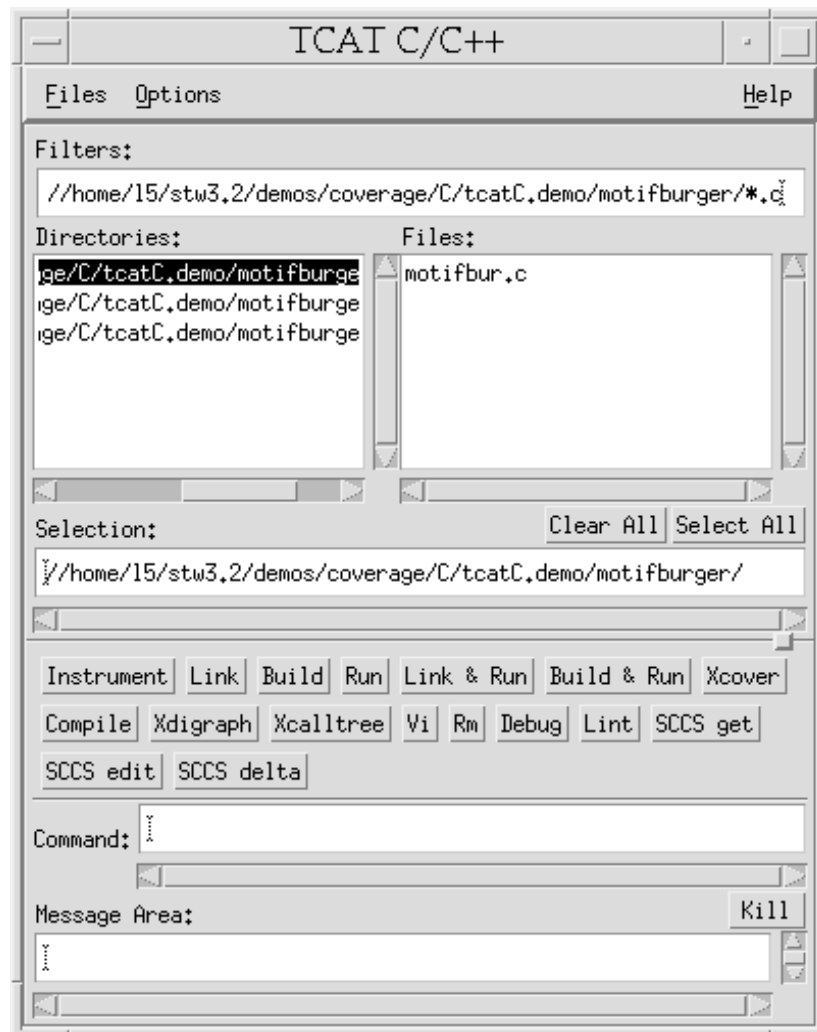
**FIGURE 2** Setting Up the Display (Initial Condition)



Now, invoke **TCAT C/C++**

1. Position the mouse pointer in the invocation window. Activate the window by clicking the mouse pointer on it.
2. To invoke **TCAT C/C++**, type in  
**xtcatt9**
3. When you type in this command, the **TCAT C/C++** main window pops up.
4. You can terminate **TCAT C/C++** at any time from the **TCAT C/C++** main window by clicking on the **Files** menu and selecting **Exit**.

When TCAT C/C++ is invoked, your display should contain this panel:



---

**FIGURE 3** The TCAT C/C++ Main Window

---

## 2.3 STEP 2: Selecting Load Setting

Selecting the load settings determines which application's resource files will be tested. The Select Load Settings menu has a filter that you can use to list \*.res files.

Select *motifbur.res*.

When you are selecting the load setting, your display should contain this panel:



**FIGURE 4** Selecting the Load Setting

## 2.4 STEP 3: Instrumenting the Example Application

After the load setting is selected, the next step is to instrument the example application. Instrumentation inserts special markers at every segment in each program module. To instrument Motifburger, use the filter in the main window to bring it up, and then select it by highlighting it in the **Files** window.

Instrument Motifburger by clicking on the **Instrument** tool button. Instrumentation can take a few seconds, during which time the tool buttons are grayed out.

---

**NOTE:** Instrumenting your application also re-compiles it.

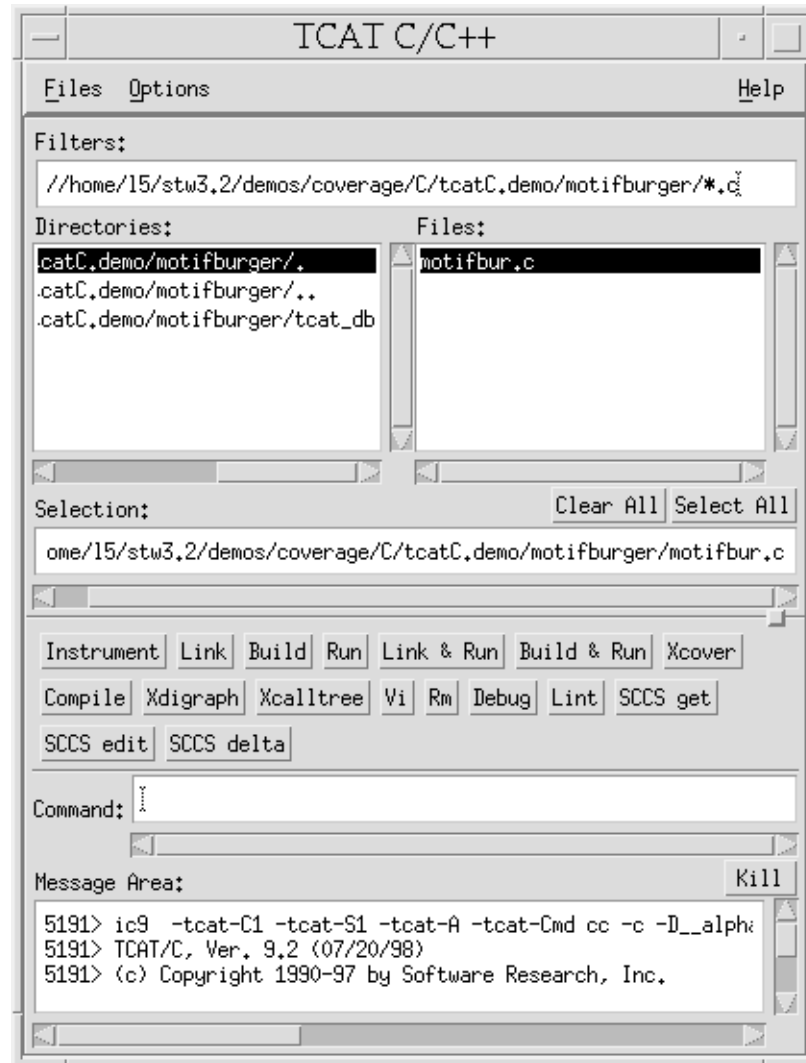
---

Instrumentation produces the following files:

- *basename.cg* — a Calltree Graph Listing.
- *basename.dg* — a Directed Graph Listing.
- *TCAT.mdf* — a database reference file.
- *basename.o* — an object file.

TCAT C/C++ puts the first three of the files listed above into the *tcat\_db* directory.

Instrumenting your application will not change its functionality. When it is linked and executed, the instrumented application will behave as it normally does, except that it will also write coverage data to a trace file.



**FIGURE 5** Instrumenting the Source File

After instrumenting the source file, enter the following command in the command line:

```
uil -o = minus sign or dash with a lower case
      "o" ( alphabet o as in on or off )
```

## 2.5 STEP 4: Choosing and Linking a Runtime Version

In this step, you specify the example runtime object module you will use to link with your instrumented application's object modules.

Software Research supplies six runtime object modules:

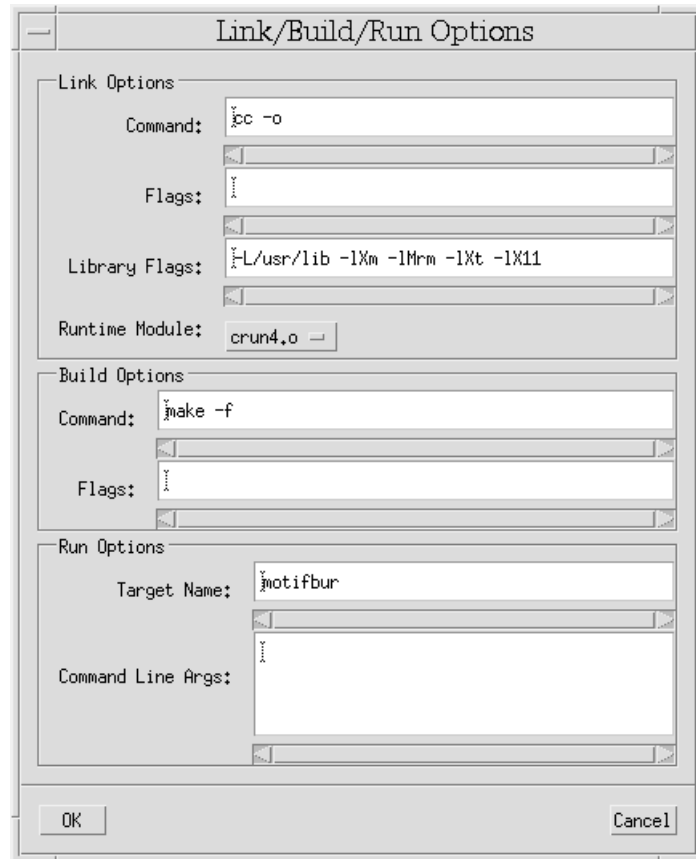
- *crun0.o*
- *crun1.o*
- *crun2.o*
- *crun3.o*
- *crun4.o* (version used most often)
- *crun5.o*

The following options are only available on certain platforms:

- *crun 4\_mt.o* (*only available on certain platforms*)
- *crun 5\_mt.o* (*only available on certain platforms*)

Each runtime object module can change the behavior and the performance of your application. To choose one of these:

1. Select the **Options** pull-down menu, and click on Link/Build/Run Options.
2. Under the Link Options heading, the Runtime Module pop-up menu allows you to specify one of the runtime object modules.
3. Select *crun4.o*
4. Click on **Okay**, and the window disappears.



**FIGURE 6** Selecting the Runtime Object Module

The next step is to link the selected runtime object module with the instrumented application's object modules, named *motifbur.o*. Linking will create an executable, because the instructions in the example program are linked to the object modules that will record program behavior during execution.

1. Use the filter to select for \*.o files. The runtime objects modules should be listed in the **Files** selection window.
2. Highlight *motifbur.o*.
3. To link, click on the **Link** button.

## 2.6 STEP 5: Running the Application

During instrumentation, TCAT C/C++ inserted function calls at each logical branch it found. In order to later determine the C1 coverage, you must run the application.

By running Motifburger and choosing the various possible combinations of hamburger meals, you are exercising segments of the Motifburger program. Because you have instrumented the program, the exercise will create trace files and allow you to view coverage information on the exercise.

To run the instrumented application:

1. Use the filter to select for the \*.out files.
2. Select the executable output file you just created when you linked.
3. Click on the **Run** tool button.

---

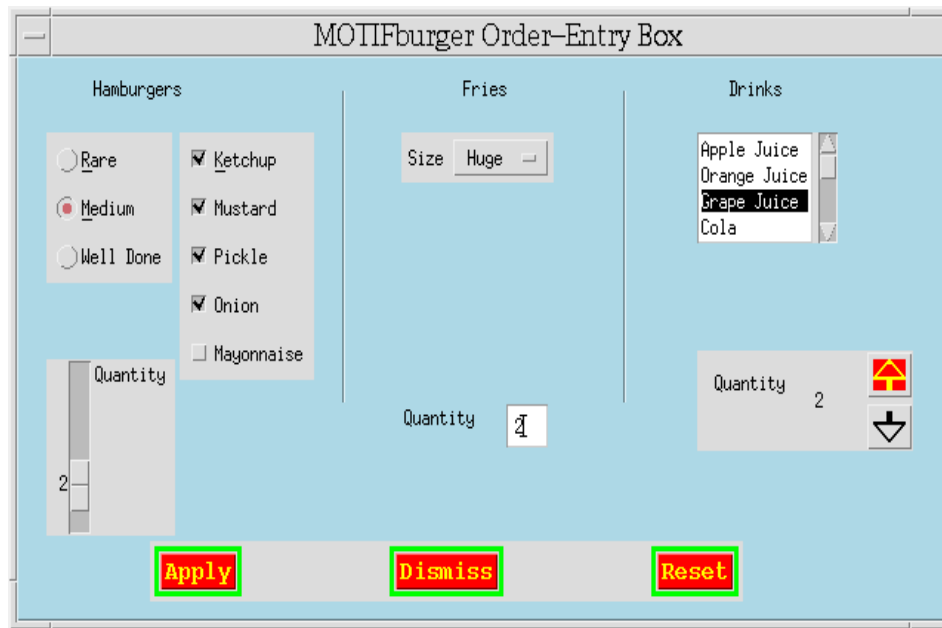
**NOTE:** An object module that is not a quiet run-time module will prompt here for a name of the tracefile and will give it a descriptor.

---

4. The Motifburger program should appear in the invocation window. Select the combination that appeals to you, specifying quantities, condiments, how the burger should be cooked, etc.
5. When you have made a selection that appeals to your virtual appetite, click the **Apply** button.



When the Motifburger is running, your display should look like this:



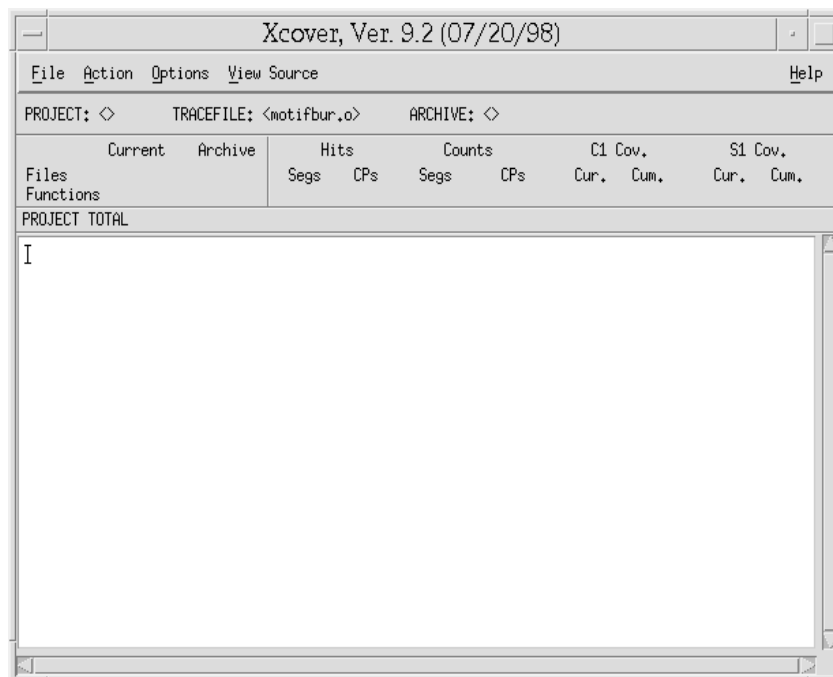
**FIGURE 7** Running Motifburger

## 2.7 STEP 6: Opening the Coverage Window

All the information from the run of the application is stored in a trace file. From the trace file, coverage reports are produced. The **XCover** window allows you look at a report, which tells you which segments have been hit.

To open the **XCover** window:

1. Click on the *TCAT C/C++* invocation window's **XCover** button.
2. The **XCover** window pops up.
3. Use the mouse to drag the window below the **TCAT C/C++** invocation window.



---

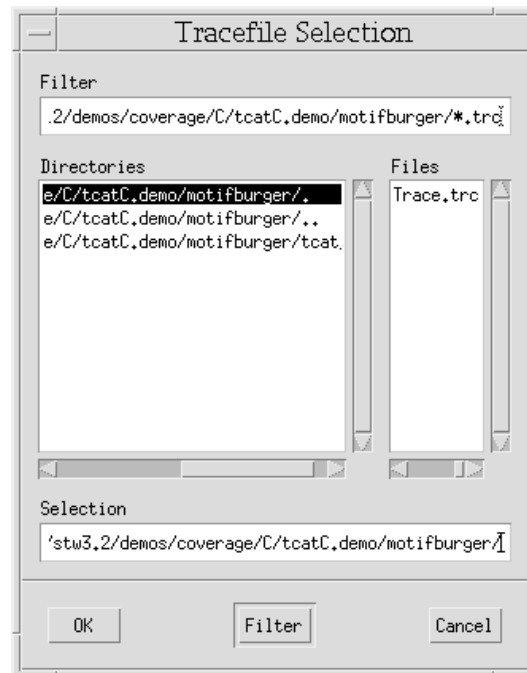
**FIGURE 8** The Coverage Window

## 2.8 STEP 6: Choosing a Trace File

Before looking at coverage reports, you must first select a trace file.

1. Click on the **File** pull-down menu.
2. Select **Trace File**.
3. A file selection dialog box pops up.
4. The Trace.trc file should be listed in the **Files** selection window.
5. Select it by clicking the mouse button on it and then clicking on **OK**. You can also highlight or type in the file name, then click on **OK** or press the <ENTER> key.

When you are selecting a trace file name, your display should look like this:



**FIGURE 9** Selecting a Trace File Name

## 2.9 STEP 8: Generating and Viewing a Report

The report must be generated and formatted before its results can be seen.

1. Select the Action pull-down menu.
2. Select Generate Report.
3. Click on the <motifbur> line to expand the display to include information on each function.

There are several fields in the report, with the following meanings:

<b>Hits</b>	The number of times the element was executed during the test.
<b>Count</b>	The total number of segments within the function.
<b>C1</b>	The percentage of branch coverage for each function.
<b>S1</b>	The percentage of call pair coverage for the function.

PROJECT: (NONE)		PROFILE: (Trace, Inc)		HITS		COUNTS		C1 Cov.		S1 Cov.	
File	Module	Current	Archive	Segs	Cps	Segs	Cps	Cur.	Dns.	Cur.	Dns.
<b>PROJECT TOTAL</b>				81	83	302	291	17.49	17.49	21.85	21.85
C:\home\J\stetD				85	88	101	97	82.48	82.48	84.95	84.95
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_sut				4	12	7	25	57.14	57.14	80.00	80.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_inc				5	6	5	6	100.00	100.00	100.00	100.00
Segment 1				1							
Segment 2				29							
Segment 3				1							
Segment 4				0							
Segment 5				1							
Call pair 1					1						
Call pair 2					1						
Call pair 3					1						
Call pair 4					1						
Call pair 5					1						
Call pair 6					1						
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_s_k				0	0	1	1	0.00	0.00	0.00	0.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_set				1	1	1	1	100.00	100.00	100.00	100.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_get				1	1	1	1	100.00	100.00	100.00	100.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_set				1	1	1	1	100.00	100.00	100.00	100.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_upd				1	2	1	2	100.00	100.00	100.00	100.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_res				3	4	3	4	100.00	100.00	100.00	100.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_c1a				0	0	1	1	0.00	0.00	0.00	0.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_act				17	25	35	35	48.57	48.57	71.43	71.43
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_top				1	0	1	0	100.00	100.00	100.00	100.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_llc				1	2	1	2	100.00	100.00	100.00	100.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_sca				1	0	1	0	100.00	100.00	100.00	100.00
C:\demo\coverage\C\ccatC_demo\ver1\Fbanger\ver1\Fbar_dho				2	2	5	5	66.67	66.67	66.67	66.67

**FIGURE 10** Coverage Information for Each Function

You can look at the same information for each segment within a function.

**4.** Click on the `<main>` line to expand the display.

Each segment within the function is displayed, along with its coverage information.

## 2.10 STEP 9: Viewing a Logical Branch's Source Code

With the display expanded to show segments, you can view the source code.

5. Click on the **View Source** pull-down menu.
6. A **View Source** window pops up.
7. Move the **View Source** window beside the **Xcover** window.
8. For this demonstration, take a look at the source code for Segment 4. To do so, position the mouse pointer on Segment 4 and press the mouse button.
9. **TCAT C/C++** automatically locates the source code for Segment 4 and displays it in the **View Source** window.
10. Use the **View Source** scroll bars to move up/down or side/side.
11. When you are finished looking at the source code, click on **View Source's Action** pull-down menu and select **Exit**. The window closes.

When **Xcover** shows the source code, your display should look like this:

```

232
233 static int init_application()
234 {
235     int k;
236
237     /* Initialize the application data structures. */
238     for (k = 0; k < MAX_WIDGETS; k++)
239         widget_array[k] = NULL;
240     for (k = 0; k < NUM_BOOLEAN; k++)
241         toggle_array[k] = FALSE;
242
243     /* Set the medium 'hamburger doneness' toggle button so that the
244      * radio box has one toggle button ON at startup. */
245     toggle_array[k_burger_medium - k_burger_min] = TRUE;
246
247     /* Initialize current values of various items to match their initial values
248      * in the DECburger UIL module. */
249     current_drink = XmStringLtoRCreate("Apple Juice","");
250     current_fries = XmStringLtoRCreate("Medium","");
251
252     /* Set up the compound strings that we need. */
253     latin_create = XmStringLtoRCreate("Create order box...", "");
254     latin_dismiss = XmStringLtoRCreate("Dismiss order box...", "");
255     latin_space = XmStringLtoRCreate(" ", "");
256     latin_zero = XmStringLtoRCreate(" 0 ", "");
257
258 }
  
```

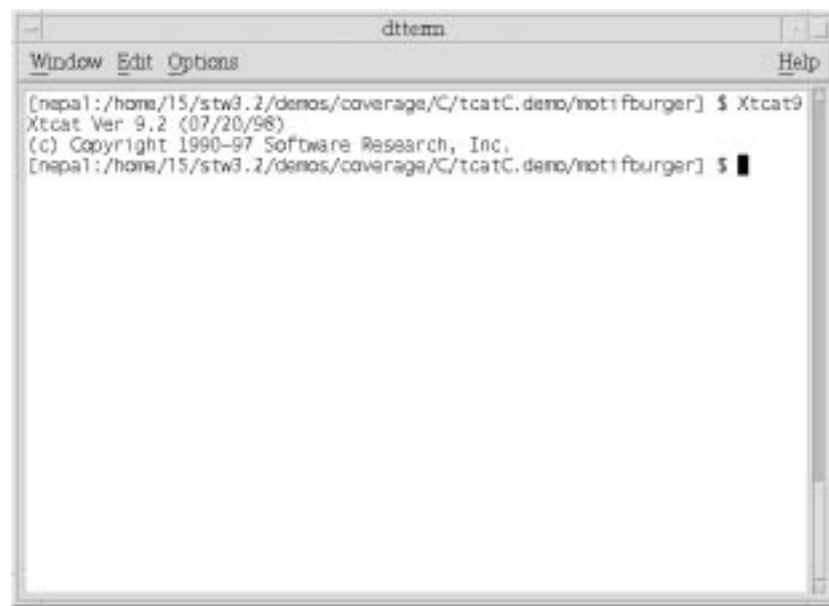
**FIGURE 11** Looking at Source Code

## 2.11 STEP 10: Sign Off and Cleanup

After looking at the source code, follow these steps to complete the session:

1. Close the **Xcover** window by clicking on the **File** pull-down menu and selecting **Exit**. You will be asked whether the archive file should be updated. Select Yes. Enter "Archive" at the end of the pathname displayed in the Save Archive dialogue box.
2. Click **File**, and **Exit**. You are asked whether you really want to exit **Xcover**. Select Okay.
3. Close the **TCAT C/C++** invocation window by clicking on the **File** pull-down menu and selecting **Exit**.

At the end of your test session, your display should look like this:



**FIGURE 12** Completing a TCAT C/C++ Session

## **2.12 Summary**

If you successfully completed the preceding steps, you've seen and practiced the basic skills you need to use **TCAT C/C++** productively. You should have learned how to invoke **TCAT C/C++**, how to instrument, compile, link, and run a program, and how to look at a coverage report.



# TCAT C/C++

## Graphical User Interface

This section describes the new TCAT C/C++ graphical user interface (GUI). This chapter applies to the Standard and Professional editions of the product.

---

### 3.1 TCAT C/C++

The new **TCAT C/C++** graphical user interface (GUI) combines the functionality of earlier releases of **TCAT** and **S-TCAT**. This new GUI allows you to instrument, compile, link, run, test, debug, build and edit your application, as well as providing point-and-click access to reports, directed graphs and calltrees.

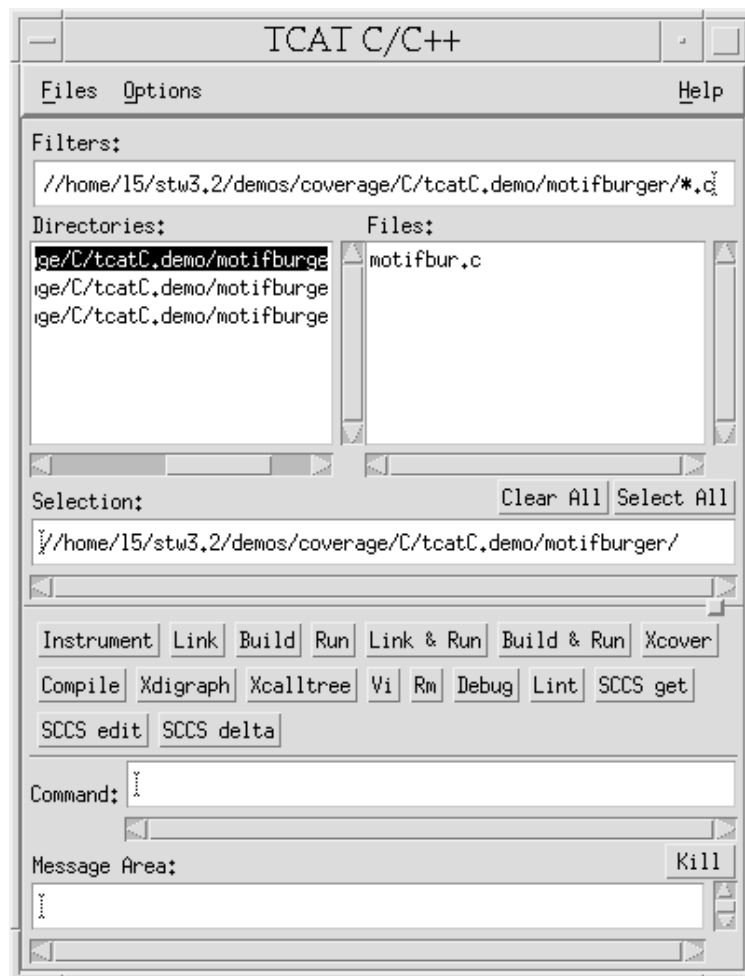
### 3.2 Invoking TCAT C/C++

**TCAT C/C++** can be invoked in one of two ways. It can be invoked from the command line with the command

```
xtcatt9
```

It can also be invoked from the STW suite's main screen, by clicking the **TCAT C/C++** icon.

### 3.3 TCAT C/C++ Main Window



---

**FIGURE 13** TCAT C/C++ main window

**3.3.1 File**

This menu allows you to manipulate the project and setting information for this session.

**3.3.2 Options**

This menu allows you to set various control settings to instrument, compile, link, build and run your application.

**3.3.3 File Selection Section**

This section of the GUI allows you to select the desired files for use with TCAT C/C++.

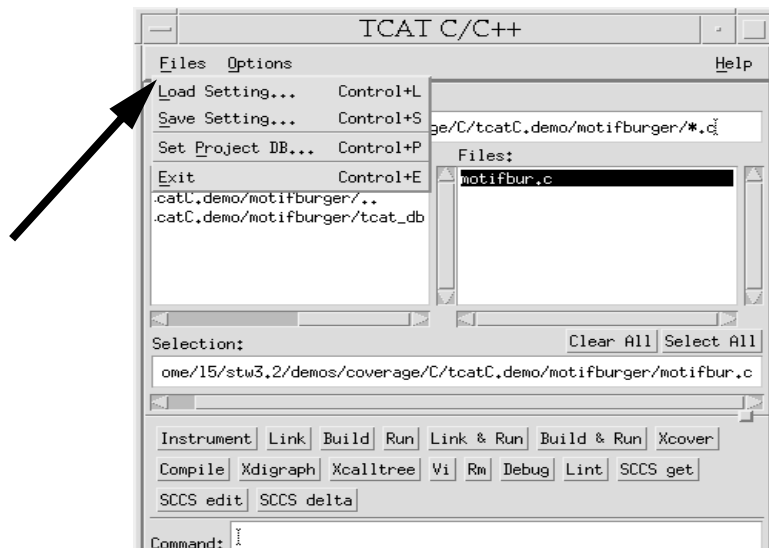
**3.3.4 Utilities Section**

This section of the GUI allows you to select various utilities to manipulate, view, and test your application

**3.3.5 Shell Section**

This section allows you to interact directly with the shell, both displaying messages from applications started from the Utilities Section, and allowing you to enter commands.

### 3.4 File Pull-Down Menu



---

**FIGURE 14** File pull-down menu

**3.4.1 Load Setting**

The option allows you to load previously saved settings for the various TCAT C/C++ options.

**3.4.2 Save Setting**

The option allows you to save settings for the various TCAT C/C++ options, so that you may load them for later use.

**3.4.3 Set Project DB**

To select a database name for this project, select this option.

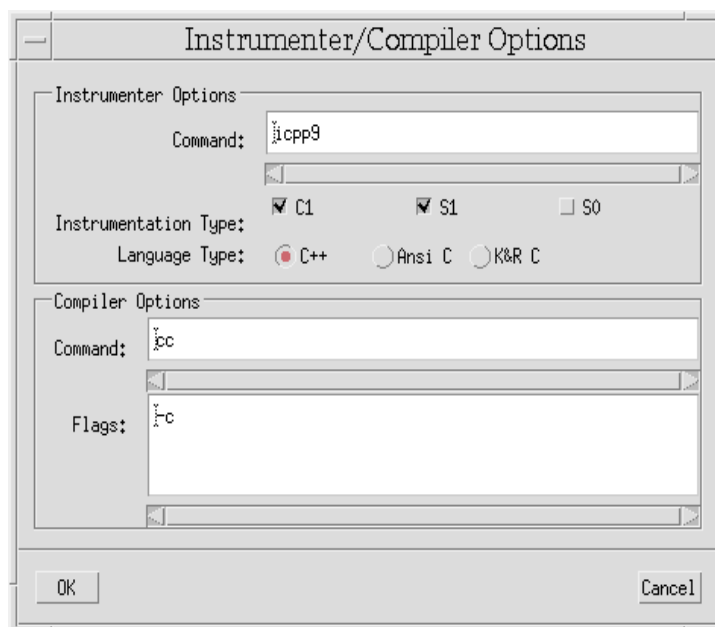
**3.4.4 Exit**

To exit TCAT C/C++, select this option.

## 3.5 Options Pull-Down Menu

### 3.5.1 Instrumentor/Compiler Options

This window allows you to choose the instrumentor and compiler to use with your application.



---

**FIGURE 15** Instrumentor/Compiler Options window

**3.5.1.1 Instrumentor Options****Command**

Enter either `ic9` or `icpp9` to select the appropriate instrumentor.

**Instrumentation Type**

This option allows you to select C1 (branch coverage) instrumentation or S1 (call pair coverage) instrumentation.

**Language Type**

This option allows you to select the language in which the target application is written.

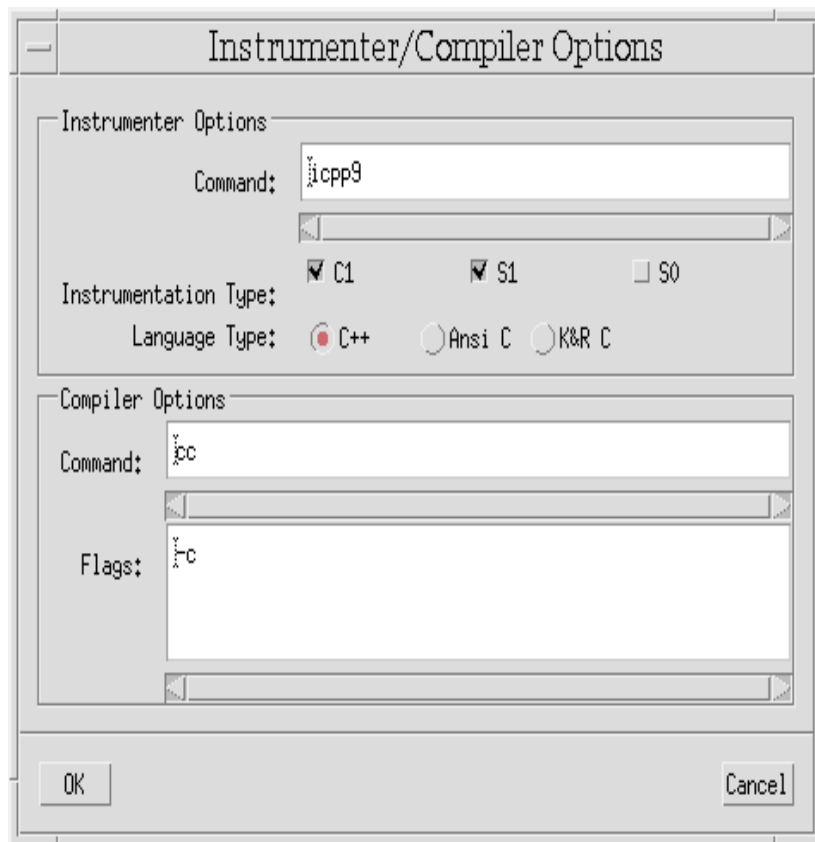
**3.5.1.2 Compiler Options****Compiler**

Enter the name of the desired compiler.

**Flags**

Entries in this field are sent to the chosen instrumentor or compiler when it is invoked.

### 3.6 Link/Build/Run Options



**FIGURE 16** Link/Build/Run Options window



**3.6.1      Link Options****3.6.1.1      Command**

This field specifies the compiler used when the target application is linked.

**3.6.1.2      Flags**

Entries in this field are sent to the compiler as flags when the target application is linked.

**3.6.1.3      Library Flags**

Entries in this field are sent to the compiler as library flags when the target application is linked.

**3.6.1.4      Runtime Module**

This pop-up menu allows you to select the runtime module to link to the instrumented application's module.

**3.6.2 Build Options**

**3.6.2.1 Command**

This option specifies the command used when you select Build.

**3.6.2.2 Flags**

Entries in this field are sent to the compiler as flags when you select Build.

**3.6.3 Run Options****3.6.3.1 Target Name**

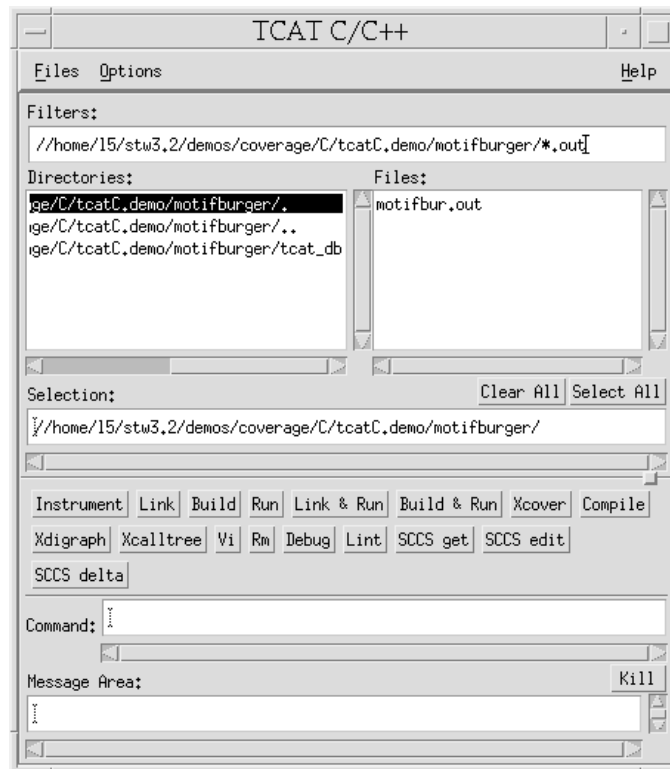
This is the name of the file created when the runtime module is linked to the instrumented application's module.

**3.6.3.2 Command Line Args**

Entries in this field are sent to the compiler as command line arguments when you select Build.

### 3.7 File Selection Section

The filter determines what files are displayed in the File box. For instance, if you were to type `*.out` in the Filters box, all files with the extension `.out` would appear in the Files box.



---

**FIGURE 17** File Selection section of TCAT C/C++ main window

**3.7.1 Directories**

This selection list displays the directories within the current directory. A scroll bar allows you to read the entire pathname.

**3.7.2 Files**

This selection list displays all the files (or a subset you have defined with a filter) in the current directory. Scroll bars allow you to see entire file names and all files in a directory.

**3.7.3 Selection**

This text field displays the current file selected, if there is exactly one. The user can also type in this field to specify the selected file.

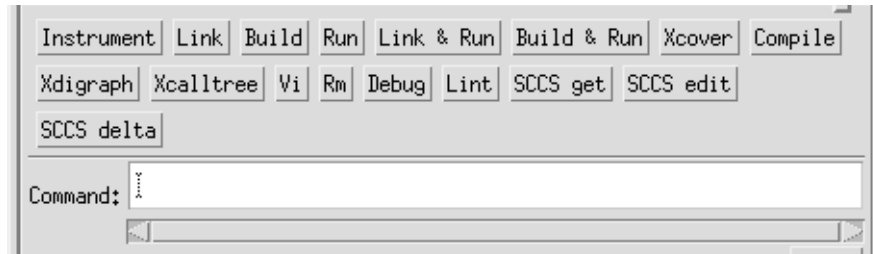
**3.7.4 Clear All**

This button clears all file selections.

**3.7.5 Select All**

This button selects all files in the Files Box.

### 3.8 Utilities Section



---

**FIGURE 18** Utilities section of TCAT C/C++ main window

**3.8.1 Instrument**

Select an application and click this button to instrument it.

**3.8.2 Link**

Having chosen a runtime module in the Link/Build/Run Option window, select the instrumented application and click this button to link it to the instrumented application's module.

**3.8.3 Build**

This option allows you to build the target application. It uses `make` files instead of object files.

**3.8.4 Run**

Select an executable created by linking, and click the Run button to execute the instrumented application.

**3.8.5 Link & Run**

This button combines the Run and Link buttons. Select the instrumented application's module and click the Link & Run button.

**3.8.6 Build & Run**

This button combines the Build and Run buttons. Select the target application, and click on Build & Run.

**3.8.7 Xcover**

This button calls up the **Xcover** utility to allow you to view coverage information about your testing. See Chapter 5 for more details.

**3.8.8      Compile**

Clicking this button compiles the selected source file with the compiler command and flags set in the Instrumentor/Compiler options window.

**3.8.9      Xdigraph**

This button calls up the **Xdigraph** utility to allow you to graphically display the directed graph corresponding to the selected file. See Chapter 7 for more details.

**3.8.10     Xcalltree**

This button calls up the **Xcalltree** utility to allow you to graphically display the calltree corresponding to the selected file. See Chapter 6 for more details.

**3.8.11     Vi**

This button calls up the **vi** editing program.

**3.8.12     Rm**

This button executes the Unix **rm** command on selected files.

**3.8.13     Debug**

This button executes the **dbx** debugger on selected files.



**3.8.14      Lint**

This button executes Lint on selected files.

**3.8.15      SCCS get**

This button executes the SCCS `get` command.

**3.8.16      SCCS edit**

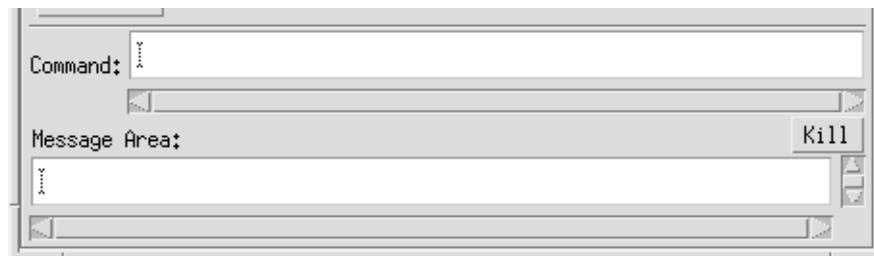
This button executes the SCCS `edit` command.

**3.8.17      SCCS delta**

This button executes the SCCS `delta` command.

The remaining buttons are defaults, but are changeable.

## 3.9 Shell Section



---

**FIGURE 19** Shell section of the TCAT C/C++ window

### 3.9.1 Command

Entries in this field will be passed onto the shell.

### 3.9.2 Message Area

Messages from the shell regarding **TCAT C/C++** are displayed in this area.

### 3.9.3 Kill

Clicking on this button will terminate any process started from the command buttons or the command text field.

# Runtime System

This chapter is a guide to TCAT C/C++ 's Runtime Options applies to all editions of the product.

---

## 4.1 TCAT C/C++ “Runtime” Support Options

Once a program is instrumented with **ic** or **icpp** it must be linked or bound with one of the various *TCAT C/C++* runtime support versions. This section describes the various versions of runtime and indicates how they affect performance of the instrumented process.

Briefly, there are two different routes you can use in setting up the runtime support for *TCAT C/C++*:

- Pre-select a set of runtime parameters and options by linking from one of the supplied set of runtimes: crunN.o, with 0, 1, 2, 3, 4, 5, 4\_mt, 5\_mt.
- Link from crunN.o and then select options by assigning parameters using an additional application-called flag **-TRACE** appearing on the command line.
- Link from crunN.o, with 0, 2, and 4 allows one to select options by assigning parameters using an alternative external environment variable:

**TCAT\_TRACE\_BUFFER\_SIZE** w/default “10,000”

**TCAT\_TRACE\_DIR** w/default “.”

**TCAT\_TRACE\_NAME** w/default “Trace.trc”

## 4.2 Pre-Selected Link-Time Options

In addition to the standard **crunN.o** (see below), for convenience, there are six pre-specified combinations of runtime support available as shown on the following table.

Level of Buffering	Trace Data Written to Trace.trc:	Asks User for	OK for Multi-tasking
None (N=1)	crun0.o	crun1.o	Yes
N-Record	crun2.o	crun3.o	Yes
Infinite (N=-1)	crun4.o	crun5.o	Yes (See Note)

---

**TABLE 1** Runtime support with fixed options

---

In the above table, the default buffering value of N is dependent on the system and may change from language to language. Typically, the default for N is set to 10,000 but it can be changed with commands in the `tcrt.rc` file (see below).

The N-record buffering should not be used when there is a chance that the instrumented process will not terminate cleanly (in which case any buffered data will be lost). During a clean termination, N-record buffering you will save all data, with the possible exception of the last N tracefile records.

If you do not expect a lot of unusual terminations then the infinite buffering, N = -1, is the most efficient (see next section).

The safest procedure of all is to use N=1, in which case only the very last tracefile record would be lost should the instrumented process not terminate cleanly.

### 4.3 Performance Gain With Buffering

There is a significant performance difference between the three levels of buffering. Here is a summary of the impact of the three different levels of runtime options on analysis of an instrumented copy of the commonly-used **xcalc** application. All execution times are based on running the same keysave file into the **xcalc** application.

Buffering Level:	Execution Time	Ratio	Percent Overhead
Uninstrumented	124.5 sec	1.0	0%
N=1	234.5 sec	2.0	100%
N=100	185,8 sec	1,9	90%
N=10000	132.6 sec	1.7	70%
Infinite (N=1)	129.5 sec	1.05	5%

TABLE 2

Comparison: Three Levels of Buffering



# *Xcover* — Coverage Analyzer

This chapter is a guide to TCAT C/C++ 's complete coverage analyzer for branch (C1) or call pair (S1) metrics, with a highly flexible graphical display. This chapter applies to all editions of TCAT C/C++.

---

## 5.1 **Xcover**

**Xcover** analyzes the trace files created when an instrumented program is executed. You can then generate and view reports based on the tracefile data using **Xcover**.

## 5.2 **Xcover Functionality**

**Xcover** makes the following assumptions:

1. A [possibly empty] archive file and a current [possibly empty] tracefile exist in the **TCAT C/C++** tracefile format.
2. There is a *tcat\_db* corresponding to any tracefiles that will be examined in the directory from which **Xcover** was opened.
3. The actual update of trace + archive —> archive is optional at the end of a session.
4. The usual rules for precedence of archive over trace prevail, and warning messages are issued when size differences between archive and tracefile are found (there should be no difference because only new-format tracefiles are processed).
5. If **Xcover** is called with no arguments, it appears on the screen and the user can select the tracefile and/or archive file to be processed.

( Assuming that there exist a *tcat\_db*. )

If there are arguments, then the data on the screen assumes that the specified Archive file and all mentioned Tracefile(s) are processed into the data on the screen.

Note that re-write of the archive file is not automatic with **Xcover**.

### 5.3 Command Line Invocation

**Xcover** is invoked from the command line with the following command. Note that the switches shown here are the ones that have the corresponding functionality with the `cover` command. **Xcover** and **cover** act the same way with regard to processing multiple tracefiles and updating the specified archive file.

`Xcover tracefile [-a archive]`

**-a archive**      Archive File Specification. This is the archive file to use. The archive file may be updated depending on user actions inside **Xcover**.

The default archive file is **Archive**.

If no archive file is given, and **Archive** does not exist in the current directory, then no archive data is used.

**-q**              Quiet option, suppress all header messages

Once Xcover appears on your display, you must generate a report from the tracefile.

1. Select the Action pull-down menu.
2. Select Generate Report.

Coverage information for the application appears.



## 5.4 Xcover Sample Screens

Following are some samples of the **Xcover** screen layout. They show analyses of the restaurant example in various stages of expansion and contraction of the display.

Xcover, Ver. 9.2 (07/20/98)

FileEditOptionsRunSource

Help

PROJECT: <TOWN>TASKFILE: <Tacas.trc>MODEM: 0

Current		Archive	Items		Counts		CI Dev.		SI Dev.			
Files	Model		Segs	OPs	Segs	OPs	Cur.	Cum.	Cur.	Cum.		
PROJECT TOTAL			83	65	363	258			17.40	17.40	21.00	21.00
C:\demo\cover\src\C\tcatC_demo\hotiflurger\hotiflur.qcd			1	2	1	2			180.00	200.00	200.00	180.00
C:\demo\cover\src\C\tcatC_demo\hotiflurger\hotiflur.res			3	4	3	4			180.00	200.00	200.00	180.00
Segment 1			1									
Segment 2			0									
Segment 3			1									
Call Item 1				8								
Call Item 2				1								
Call Item 3				1								
Call Item 4				1								
C:\demo\cover\src\C\tcatC_demo\hotiflurger\hotiflur.sla			0	0	1	1			8.00	8.00	0.00	8.00
Segment 5			0									
Call Item 5				8								
C:\demo\cover\src\C\tcatC_demo\hotiflurger\hotiflur.act			17	20	26	36			46.50	46.50	21.00	21.40
Segment 6			20									
Segment 7			0									
Segment 8			0									
Segment 9			0									
Segment 10			0									
Segment 11			0									
Segment 12			0									
Segment 13			0									
Segment 14			0									
Segment 15			1									
Segment 16			1									
Segment 17			0									
Segment 18			8									
Segment 19			1									
Segment 20			6									
Segment 21			2									
Segment 22			1									
Segment 23			0									
Segment 24			1									
Segment 25			0									
Segment 26			1									
Segment 27			2									
Segment 28			3									
Segment 29			4									
Segment 30			5									
Segment 31			5									
Segment 32			0									
Segment 33			0									
Segment 34			0									
Segment 35			0									
Call Item 6				8								
Call Item 7				8								
Call Item 8				8								
Call Item 9				8								
Call Item 10				8								
Call Item 11				8								
Call Item 12				8								
Call Item 13				8								
Call Item 14				8								
Call Item 15				8								
Call Item 16				8								
Call Item 17				8								
Call Item 18				8								

FIGURE 20 Xcover Example Output 1

Xcover, Ver. 9.2 (07/20/98)										
File Action Options View Source Help										
PROJECT: <TCAT>		TRACEFILE: <Trace.trc1>		ARCHIVE: <>						
Files	Current	Archive	Hits		Counts		C1 Cov.		S1 Cov.	
	1	0	Segs	CPs	Segs	CPs	Cur.	Cum.	Cur.	Cum.
Modules	18	0								
PROJECT TOTAL			49	63	101	97	48,51	48,51	64,95	64,95
</usr/stw3>			49	63	101	97	48,51	48,51	64,95	64,95
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,mal>			4	12	7	15	57,14	57,14	80,00	80,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,ini>			5	6	5	6	100,00	100,00	100,00	100,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,s_e>			0	0	1	1	0,00	0,00	0,00	0,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,set>			1	1	1	1	100,00	100,00	100,00	100,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,get>			1	1	1	1	100,00	100,00	100,00	100,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,set>			1	1	1	1	100,00	100,00	100,00	100,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,upd>			1	2	1	2	100,00	100,00	100,00	100,00
Segment 1			4							
Callpair 1				4						
Callpair 2				4						
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,res>			3	4	3	4	100,00	100,00	100,00	100,00
Segment 1			1							
Segment 2			8							
Segment 3			1							
Callpair 1				8						
Callpair 2				1						
Callpair 3				1						
Callpair 4				1						
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,cle>			0	0	1	1	0,00	0,00	0,00	0,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,act>			13	25	35	35	37,14	37,14	71,43	71,43
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,tog>			1	0	1	0	100,00	100,00	100,00	100,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,lis>			1	2	1	2	100,00	100,00	100,00	100,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,sca>			1	0	1	0	100,00	100,00	100,00	100,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,sho>			2	2	3	3	66,67	66,67	66,67	66,67
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,sho>			0	0	7	5	0,00	0,00	0,00	0,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,cre>			5	3	5	3	100,00	100,00	100,00	100,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,qui>			2	1	3	1	66,67	66,67	100,00	100,00
<2/demos/coverage/C/tcatC,demo/motifburger/motifbur,pul>			8	3	24	16	33,33	33,33	18,75	18,75

FIGURE 21 Xcover Example Output 2

## 5.5 Operation of Xcover

Each whole line in the display is sensitive to mouse clicks. One click expands the entry; another click contracts it.

When the report is first displayed, only the current archive and tracefile module names appear. (The syntax for naming them is identical to *cover*). The expansion/contraction sequence for the display is as follows. The default starting point of expansion is *module*name in both cases.

C1 expansion tree is: filename > modulename > segment > text of segment;

S1 expansion tree is: filename > modulename > call pair > text of call pair.

The entire display, including all expansions, is fitted into a two-dimensional scroll window.

If the View Source window is open, and you click on a segment or call pair, the appropriate source code is displayed.

### 5.5.1 Xcover Options

If the project file is absent you still get trace data but it can't be reflected back into the source. The project file is a list of file names including ./'s, ?'s, and \*'s (like UNIX filename expansion conventions) that defines the set of files that are being discussed.

### 5.5.2 Selecting a Trace File

You select a trace file using the **File** pull-down menu.

### 5.5.3 Selecting an Archive File

You select an archive file using the **File** pull-down menu.

# Xcalltree Utility

This chapter explains the Xcalltree Utility, which is a graphic display of the relationship between two called functions. This chapter applies to all editions of TCAT C/C++ .

---

## 6.1 Purpose

The **Xcalltree** utility displays the caller-callee dependence structure in a software program. The call tree is shown for the specified call pair file—the one used when you invoke **Xcalltree** — and based on files created using the **TCAT C/C++** tools.

A call pair file's relationships are annotated on the calltree, and there are ten built-in annotation options and one user-defined annotation. This information can be displayed and printed in a variety of ways.

## 6.2 Xcalltree File Format

The format for an **Xcalltree** chart file is very simple.

- # in Column 1 indicates a comment. There is no limit on the number of # comments in a file.
- The first blank line “ends the data.” This means that the information describing a calltree chart must appear before the first blank lines — and that you can have no blank lines anywhere in the data region.
- After the first blank line, the rest of the file is treated as a comment.

### 6.3 Invoking Xcalltree

**Xcalltree** can be invoked from the command line by typing:

```
Xcalltree filename  
[-D]  
[-r]  
[-m]  
[-h]
```

If you do this, the filename typed will be represented in the **Xcalltree** Main Window (see Figure 22 on page 61). The switches have the following values:

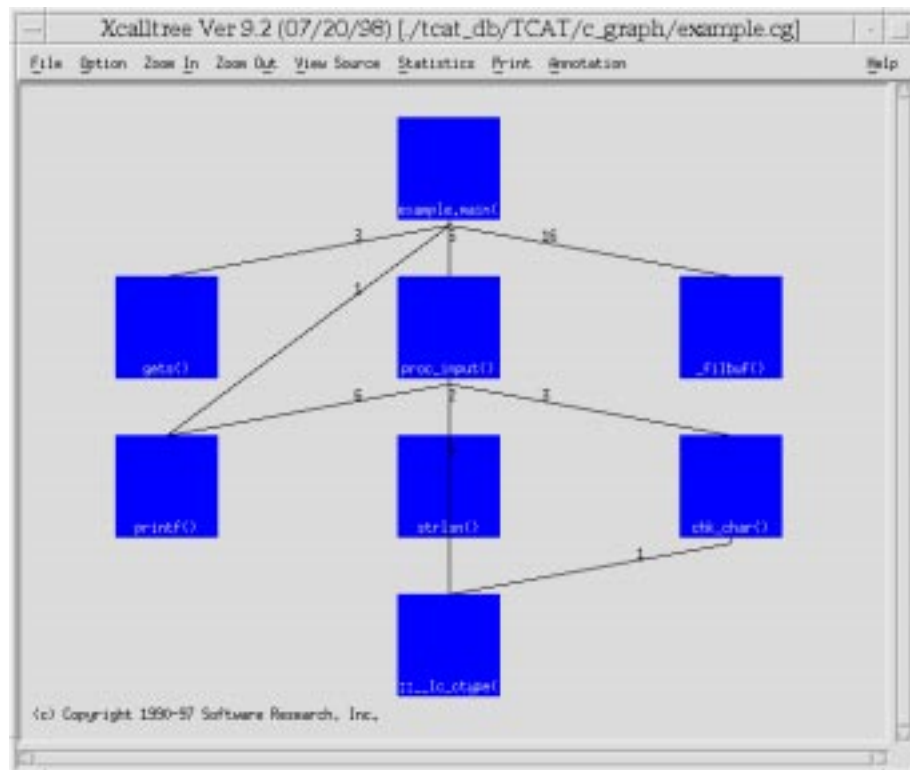
-D	Maximum depth of calltree.
-r	Rootname for top-most file of calltree.
-m	Multigraph mode.
-h	This switch brings up the <b>Xcalltree</b> help window.

You can also simply type:

```
Xcalltree
```

A blank Main Window will appear. You would then select a file name from the **File** pull-down menu.

## 6.4 Xcalltree Main Window



**FIGURE 22** Xcalltree main window

**6.4.1 File**

This pull-down menu allows you to select the file that will be displayed in the calltree.

**6.4.2 Options**

This window allows you to choose the characteristics of the nodes and edges displayed in the calltree, including shape, size, and color, as well as the scale for the **Zoom In & Zoom Out** options.

**6.4.3 Zoom In & Zoom Out**

These options allow you to expand or contract the focus of the calltree, so that you can see it in more detail or wider perspective, depending on your needs.

**6.4.4 View Source**

This window allows you to view the source code for the current calltree.

**6.4.5 Statistics**

This window allows you to display pertinent statistics about the calltree, including links, number of call pairs, calltree depth, and number of recursive modules.

**6.4.6 Print**

This window allows you to set the parameters for the calltree to be printed in your environment.



**6.4.7 Annotation**

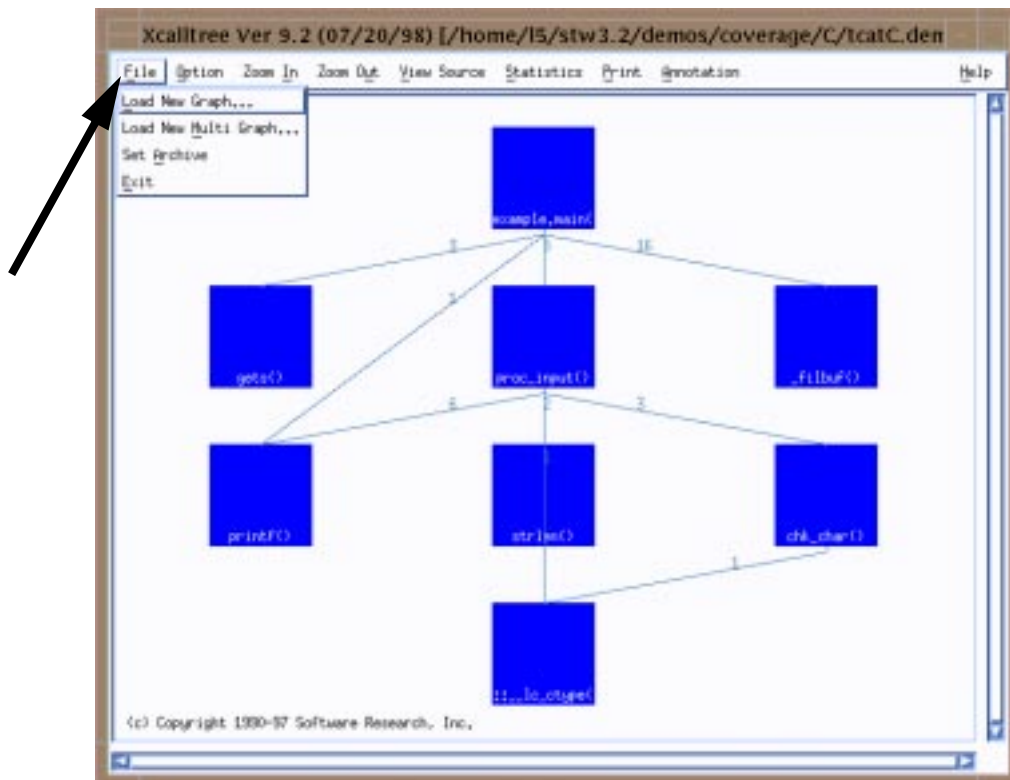
This window allows you to set the maximum and minimum thresholds for the nodes and edges in the calltree, as well as its path file.

**6.4.8 Help**

If you have a problem using **Xcalltree**, click on **Help**. Click your mouse on the **Action** pull-down menu and select **Search**. You will then get an **Enter String to search** dialog box. Click on the blank area and type the name of the option or function with which you need help.

**NOTE:** All these menus are explained in further detail on the following pages.

## 6.5 File Pull-Down Menu



---

**FIGURE 23** File Pull-Down Menu

### 6.5.1 Load New Graph

To display a calltree, click the mouse on the **File** pull-down menu. Drag the mouse to **Load New Graph**. The dialog box in Figure 24 will appear onscreen.

### 6.5.2 Load New Multi Graph

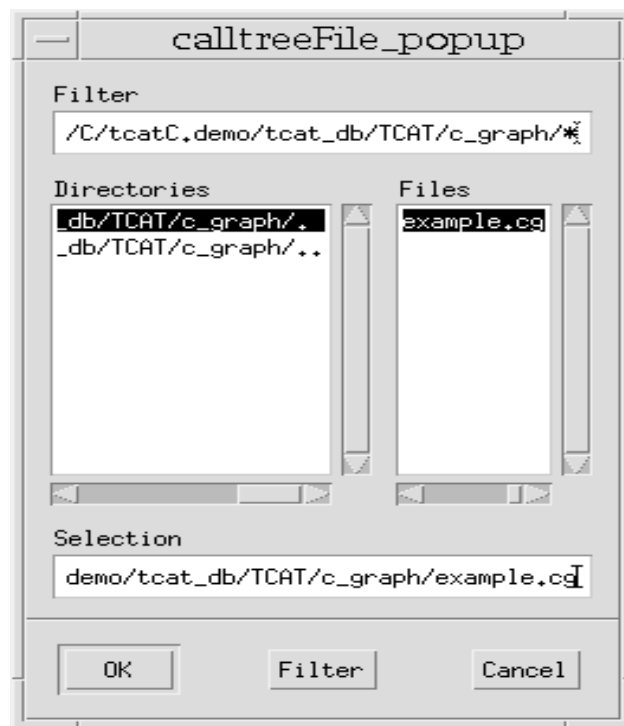
If there is more than one call between two nodes, the calltree will show *each* connection if **Load New Multi Graph** is selected. This may be difficult to see on a large calltree, but the example included in our demos directory is simple enough to see these connections.

### 6.5.3 Set Archive

Annotation of the display can be accomplished with the **Annotations** button. In many cases, annotation of the display is accomplished by showing the results of coverage testing, as reflected in the repository of multi-test coverage stored in the Archive file.

The default Archive file is "Archive," but you can change it to any file you wish using the **Set Archive** button. After you push the button you will be given a file-selection popup. Select the file you want to use as the Archive file and click on **Apply** to confirm that choice. The current name of the Archive file is shown in the filename section of the window.

## 6.6 Calltree File Selection Dialog Box



**FIGURE 24** Calltree File Selection Dialog Box

This window pops up after you select **Load New Graph** or **Load New Multi Graph**, and allows you to select the file to be displayed as a call-tree, using seven options.

**6.6.1 Filter**

Allows you to limit the number of files that will be searched for; only those ending in **.cg** will be included.

**6.6.2 Directories**

The directory from which the file to display in the calltree is chosen. Click on the chosen directory; it will be highlighted on the screen.

**6.6.3 Files**

The actual file name selected to display in the calltree. Double-click to choose, and the choice will be displayed in the **Selection** box.

**6.6.4 Selection**

Displays the file name selected in the **Files** box, or you can type in another name.

**6.6.5 OK**

Clicking on the **OK** button will cause whatever file is currently in the **Selection** box to be displayed in the calltree.

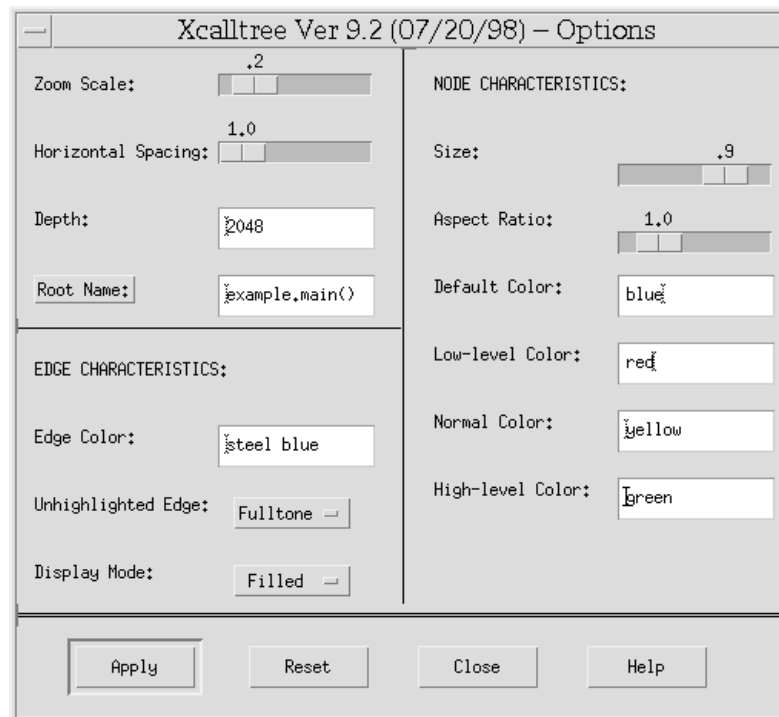
**6.6.6 Filter Button**

This button activates whatever filtering has been specified in the **Filter** box at the top of the window.

**6.6.7 Cancel**

To close the **File** dialog box without selecting a file for display, simply click on the **Cancel** button.

## 6.7 Option Window



---

**FIGURE 25** Option window

### 6.7.1 Zoom Scale

The percentage for the **Zoom In** and **Zoom Out** functions. The default setting is 0.2, which means there will be a 20% enlargement or reduction. This value can be changed by sliding the ruler to the left (smaller) or right (larger). Each 0.1 is equal to 10%, so that setting the ruler to 0.4 would mean a 40% reduction or enlargement of the calltree each time you click **Zoom In** or **Zoom Out**.

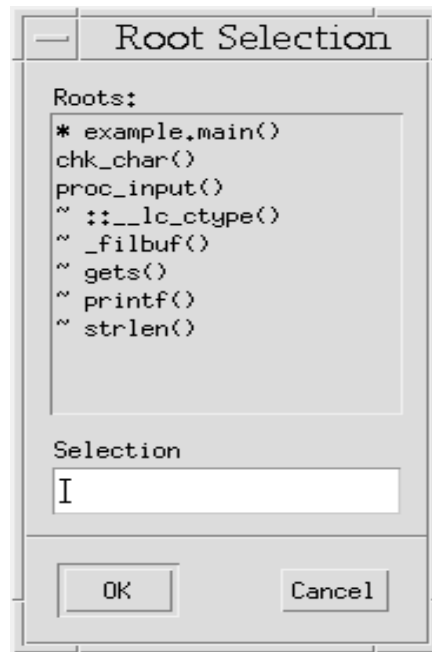
### 6.7.2 Horizontal Spacing

The space between the nodes in the calltree. The default setting is 1.0.

### 6.7.3 Depth

The **Depth** value specifies the number of layers of the tree that will be displayed. The default value, 2048, is very large and it is unlikely that any real-world calltree will be that deep. You would set the value to a smaller number, e.g. 10, if you wanted to limit the amount of detail on the screen. Using a smaller value for depth tells **Xcalltree** to disregard *all* calls below the specified value.

Also note that the **Connections** option can be adjusted to have a maximum upward and downward extent.

**6.7.4 Root Name**

---

**FIGURE 26** Root Name Selection window example 1

The call tree on the display is normally the first one occurring in the callpairs file that **Xcalltree** processes. Some call pairs files contain more than one tree, i.e., more than one single “root” module name and the associated calls. If you want to view a different calltree than the one on the display, you do so by clicking on the **Root Name** button.

The resulting root-selection window is shown in Figure 27 on page 71. Every possible function name is shown in the list in the floating window.

Modules that are possible “roots” for the call tree, i.e. which are not called by another name in the file, are shown with a “\*”. These are shown in alphabetical order at the top of the list.

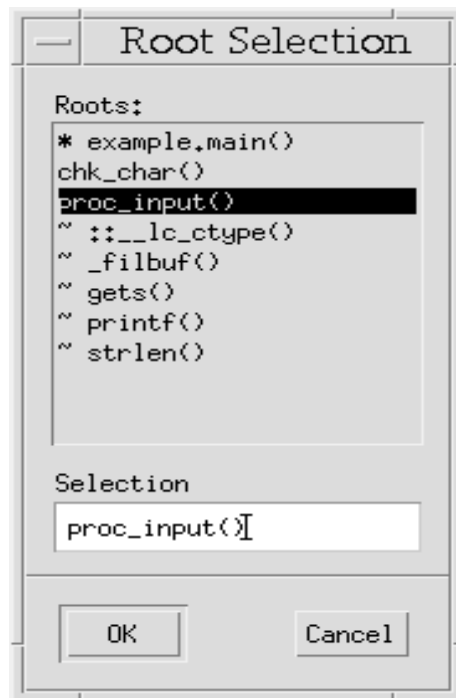
Modules that NEVER call another module are shown with a “~” in front of the name (as in Figure 26). They are sorted to the bottom of the list.

All other modules, those which are called by some root name or are in the downward chain from some root — any one of which could be chosen as a new “root” name — are in the middle of the list. Simply click on the



name you wish to be the root. The new call-tree using that name is shown.

**NOTE:** If the Depth Value is set to a low number only PART of a tree may be visible.



**FIGURE 27** Root Name Selection Window Example 2

**6.7.5      Edge Characteristics**

**6.7.5.1      Edge Color**

The actual color of the edge. Default setting is **steel blue**.

**6.7.5.2      Unhighlighted Edge**

The kind of unhighlighted edge to use: **Fulltone**, **Halftone** (dashes), or **Blank** (no lines). Default setting is **Fulltone**.

**6.7.5.3      Display Mode**

Determines whether the nodes are darkened (**Filled**) or outlined (**Outline**). Default setting is **Filled**.

**6.7.6 Node Characteristics****6.7.6.1 Size**

The relative size of the box representing each nodule. Boxes are used for “normal” functions. Circles are used for self-referencing modules. Triangles are used for modules that are invoked recursively.

**6.7.6.2 Aspect Ratio**

The height-to-width ratio of the box.

**6.7.6.3 Default Color**

Selects the basic color of the calltree's edges and nodes. The default setting is **blue**.

**6.7.6.4 Low-level Color**

In all cases, if the value of the chosen annotation is below the values indicated for Threshold 1, the display is done in the Low-level color. The default setting is **red**.

**6.7.6.5 Normal Color**

If the value of the chosen annotation is between Threshold 1 and Threshold 2, the Normal color is used (only when some edges are highlighted). The default setting is **yellow**.

**6.7.6.6 High-level Color**

If the value of the chosen annotation is above the value stated in Threshold 2, then the High-level color is used. The default setting is **green**.

---

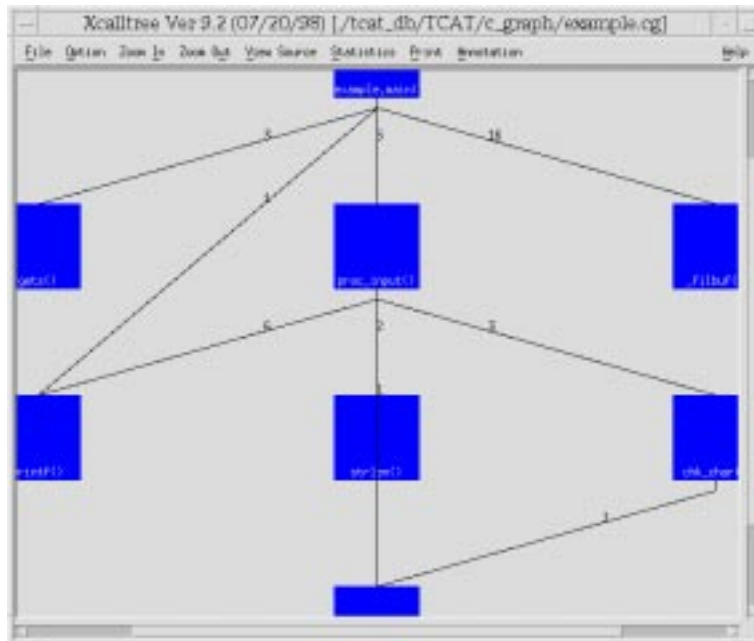
**NOTE:** If you have a monochrome display, then the three colors are expressed as a narrow, normal, and triple-wide line.

---

**6.7.6.7 Apply**

You must click on the **Apply** button in order for the new settings to be applied.

## 6.8 Zoom In & Zoom Out Options



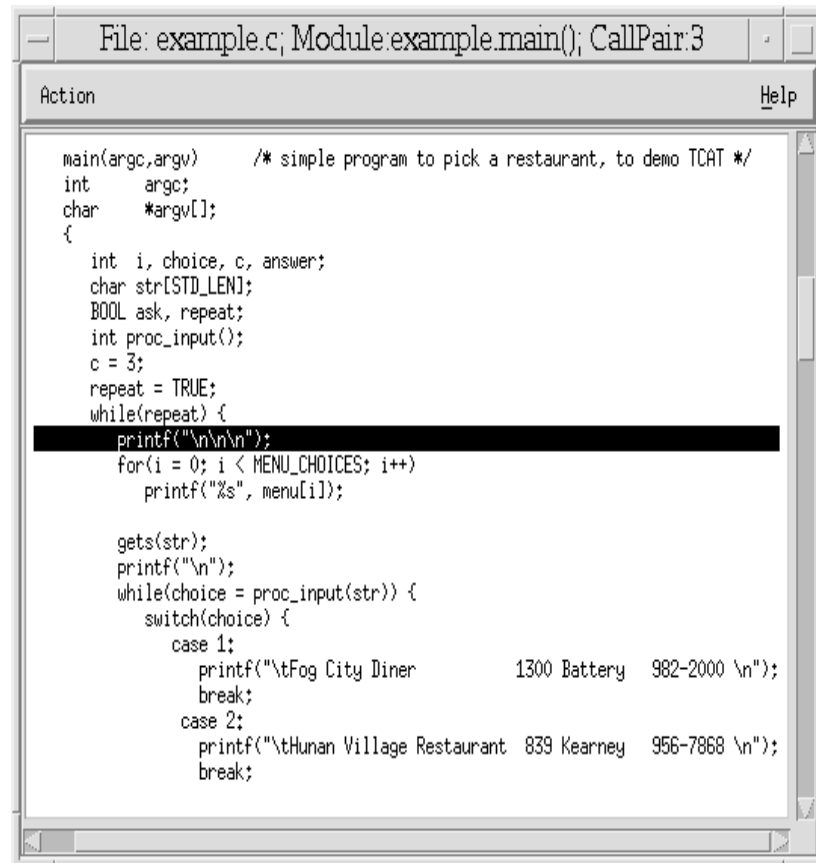
**FIGURE 28** Zoom In Option illustrated

The zoom buttons allow for a narrower or wider perspective of the calltree, depending on what you require. Click on the **Zoom In** button once to narrow the focus of the calltree, and click on the **Zoom Out** button to get a wider perspective of the calltree. Notice the difference between the calltree in Figure 28 on page 74, after clicking on **Zoom In** once, and the same calltree, depicted in Figure 22 on page 61.

The white arrow (triangle) symbols on black background on the right-hand side and bottom of the window are scroll bars, which you can use to move vertically or horizontally in viewing the calltree. You can single-click the mouse as many times as necessary to get to the desired viewing point, or for quicker response simply click and hold the mouse down.

**Note:** This feature is limited by your machine's display capabilities.

## 6.9 View Source Window



```
File: example.c; Module: example.main(); CallPair: 3
Action Help

main(argc,argv)      /* simple program to pick a restaurant, to demo TCAT */
int    argc;
char   *argv[];
{
    int i, choice, c, answer;
    char str[STD_LEN];
    BOOL ask, repeat;
    int proc_input();
    c = 3;
    repeat = TRUE;
    while(repeat) {
        printf("\\n\\n\\n");
        for(i = 0; i < MENU_CHOICES; i++)
            printf("%s", menu[i]);

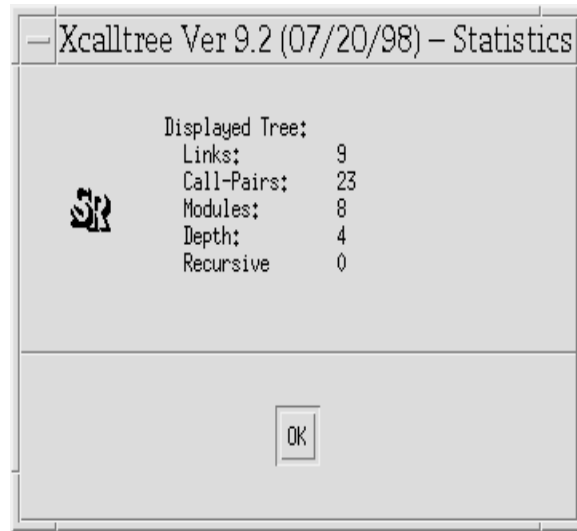
        gets(str);
        printf("\\n");
        while(choice = proc_input(str)) {
            switch(choice) {
                case 1:
                    printf("\\tFog City Diner      1300 Battery   982-2000 \\n");
                    break;
                case 2:
                    printf("\\tHunan Village Restaurant  839 Kearney   956-7868 \\n");
                    break;
            }
        }
    }
}
```

**FIGURE 29** View Source Window

### 6.9.1 Description of Source Code Viewing

The source-code text you see corresponds to the directed graph. The text is positioned to show you the location of the call pair you clicked on (or the first call pair in the module, if you don't have a multi-graph on the screen). Also, if you click on the name of a function, **Xcalltree** will invoke **Xdigraph** and show you the detailed structure of that function. From **Xdigraph** you can view the source from that perspective, i.e., in terms of edges and nodes rather than call pairs.

## 6.10 Statistics Window



**FIGURE 30** Statistics Window

The statistics you are given by **Xcalltree** are in two sections, the first pertaining to the calltree that you see on the screen, and the second pertaining to the entire file of information you supplied to the call to **Xcalltree**.

**6.10.1 Links**

This is the number of module-to-module connections in the diagram.

**6.10.2 Call pairs**

The total number of distinct, individual caller-to-callee connections in the diagram.

**6.10.3 Modules/Depth**

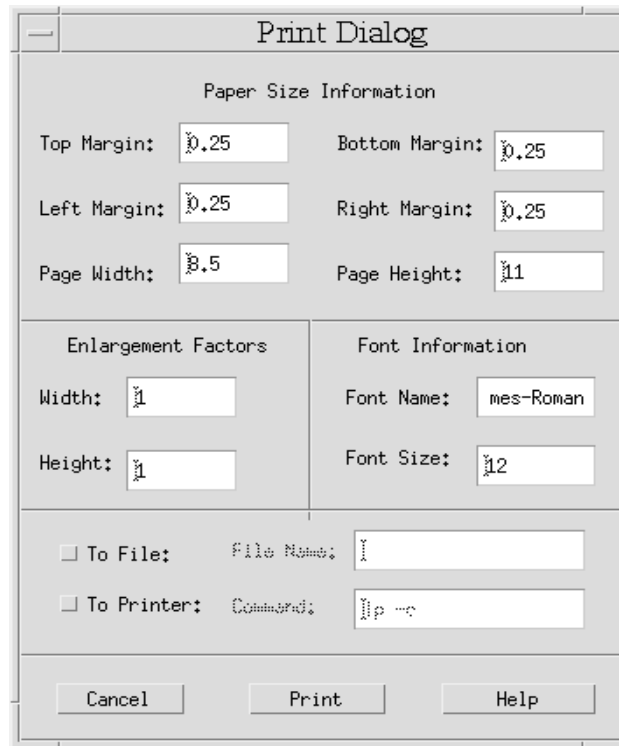
Modules is the total number of different names in the calltree, and depth indicates the maximum depth (either for links or for calltree pairs).

**6.10.4 Recursive**

If the calltree is recursive, that is, if some module calls itself or calls some other module for which there is a “self-referencing” chain, the number of such functions will be shown here.



## 6.11 Print Window



**FIGURE 31** Print Window

The image you see will be printed to a standard print device. This window will allow you to configure the printing for your environment.

**6.11.1 Paper Size Information**

<b>Top Margin</b>	The distance from the top of the page to the first line. Default setting is 0.25 inches.
<b>Left Margin</b>	The distance from the left-hand side of the page to the first character of type. Default setting is 0.25 inches.
<b>Page Width</b>	The actual horizontal width of the paper you will be printing on. Default setting is 8.5 inches.
<b>Bottom Margin</b>	The distance from the bottom of the page to the last printed line. Default setting is 0.25 inches.
<b>Right Margin</b>	The distance from the right-hand side of the page to the last character on the line. Default setting is 0.25 inches.
<b>Page Height</b>	Actual vertical length of the paper to be printed. Default setting is 11 inches.

**6.11.2      Enlargement Factors**

The enlargement factors specify the size expansion, vertically or horizontally, to be applied to this particular print activity — in effect, the total number of 8.5 inch by 11.0 inch sheets on which to draw the picture.

Selecting 1.0 means the picture will be kept on a single 8.5 inch x 11.0 inch sheet. Hence, width = 1.0 and height = 1.0 will fit the image on a standard page.

If you change the width to 2.0, however, the picture will be drawn on two pages, in such a way that two 8.5 inch by 11.0 inch sheets can be pasted together to make a 17.0 inch by 11.0 inch image. When more than one sheet is involved, the software numbers each page (on the bottom center) so that assembly into a larger diagram is simple and straightforward. To assemble a diagram, start with sheet #1 in the lower left-hand corner.

The software automatically sizes the image to fit into the smallest whole number of page equivalents. Also, the software sizes the diagram and the typefaces to “best fit” the specified size.

Some experimentation may be required to determine the optimum size for a diagram.

---

**NOTE:** The picture drawn on the printer always includes all of the information in the diagram, even if the entire diagram is not visible because of a zoom setting.

---

**6.11.3 Font Information**

The default font size, 12 point, and the default font name, Times-Roman, normally provide good quality pictures. Times-Roman at 12 point is commonly available on most printers.

You can choose different typesizes and type fonts depending on the sizes and fonts available on your computer.

**6.11.4 Print Locator**

**To File** Will create a PostScript (.ps) file, which you can use to have the calltree printed on any PostScript-compatible printer.

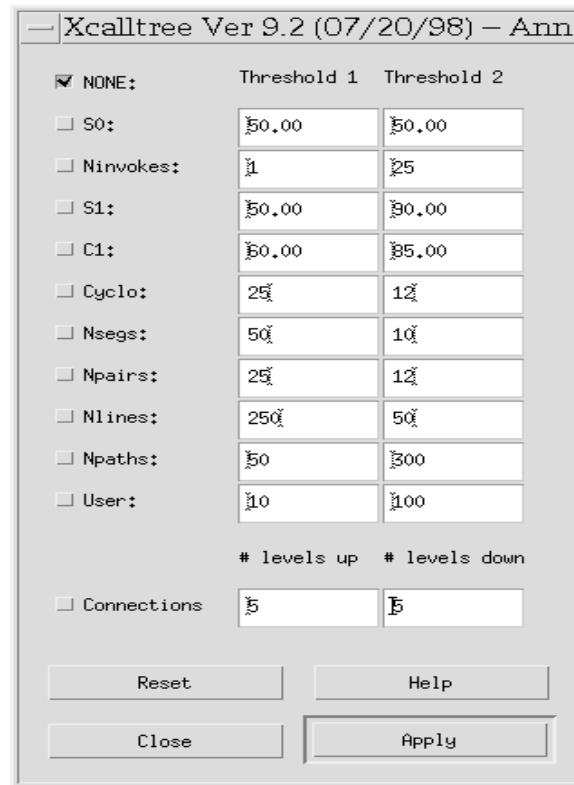
**To Printer** You must name the printer to which the printing of the document will be sent.  
When a print job has been sent to either a .ps file or to a printer, a message window saying **Print action completed** will pop up. Click **OK** to close this window.

---

**NOTE:** The print option requires a PostScript-compatible printer. If your machine is not attached to a PostScript compatible printer then the Print window option will be inoperative.

---

## 6.12 Annotation Window



**FIGURE 32** Annotation Window

There are a number of ways to annotate the calltree. Typically this involves choosing a different color depending on where a particular parameter falls into user-specified ranges (thresholds).

There are ten built-in annotation options and one user-defined annotation.

**6.12.1      Threshold 1 & Threshold 2**

Threshold 1 represents the lower limit, and Threshold 2 the upper limit desired for each metric. You can change the values of any threshold used in the annotation of the call tree by clicking in the window and typing in the new value. The values WON'T be applied to the current calltree unless you click the **Apply** button.

**6.12.2      None**

No annotation is shown.

**6.12.3      S0**

The current value of the S0 metric is used to color the display.

**6.12.4      Ninvokes**

The current number of invocations of the module is used to color the display.

**6.12.5      S1**

Call pair coverage. The current value of the S1 (module coverage) metric is used to color the display.

**6.12.6      C1**

Branch coverage. The current value of the C1 (module coverage) is used to color the display.

**6.12.7      Cyclo**

Cyclomatic complexity. The value of the cyclomatic complexity is used to color the display.

**6.12.8      Nsegs**

The number of segments in the module is used to color the display.

**6.12.9 Npairs**

The number of call pairs in the module is the metric used to color the display.

**6.12.10 Nlines**

Number of source lines. The number of non-blank lines in the module is the metric used to color the display.

**6.12.11 Npaths**

The number of paths in the selected module, as computed by `apg`, is the value used to color the display.

**6.12.12 User**

User-defined function. The outcome of calling a user-defined function, “`Xcalltree.user`”, if it exists, is the value used to color the display.

**6.12.13 Connections**

The **Connections** option can be adjusted to have a maximum upward and downward extent.

**6.12.14 Apply**

After setting the desired thresholds, click **Apply** to display the current calltree with them.

**6.12.15 Reset**

To restore the default settings to the window, click on **Reset**.

**6.12.16 Close**

To exit the **Annotation** window, click on **Close**.

**6.12.17 Help**

If you have a problem using the **Annotation** window, click on **Help**. Click your mouse on the **Action** pull-down menu and select **Search**. You will then get an **Enter String to search** dialog box. Click on the blank area and type the name of the option or function with which you are experiencing difficulty.

---

**NOTE:** When annotating the calltree, you may attempt to annotate an object file that is supplied through *X* or the machine language, to which you will not typically have the source code. When you click on a module of this type, a message box will pop up telling you that the module is not defined in the reference file.

---



### 6.13 Quick Reference Guide to Xcalltree Annotations

Function	Display Coloring Reflects What Information?	Preset Low/High
S0	Whether or not module was invoked from <b>Archive</b> file. Shows only two colors on display, low and high. The <b>Archive</b> file must be from <i>TCAT C/C++</i> . If not, silence.	50.00 / 50.00
Ninvokes	Number of times module was invoked, from <b>Archive</b> file. The <b>Archive</b> file must be from <i>TCAT C/C++</i> . If not, silence.	1 / 25
S1	S1 value for module, from <b>Archive</b> file. Module name must appear in <b>Archive</b> ; else no default color. The <b>Archive</b> file must be from <i>TCAT C/C++</i> . If not, silence.	50.00 / 90.00
C1	C1 value for module, from <b>Archive</b> file. Assumes module name appears in <b>Archive</b> ; else no color. The <b>Archive</b> file must be from <i>TCAT C/C++</i> . If not, silence.	60.00 / 85.00
Cyclo	Cyclomatic complexity.	12 / 25
Nsegs	The number of segments in the module. Requires use of <i>TCAT C/C++ Ver 9</i> .	10 / 50
Npairs	Number of call pairs in module, from <b>tcats_db</b> . The <b>tcats_db</b> must be from <i>TCAT C/C++</i> . If not, silence.	12 / 25
Nlines	Number of source lines. The number of non-blank lines in the module, from <b>tcats_db</b> . The <b>tcats_db</b> must be from <i>TCAT C/C++</i> . Requires use of <i>TCAT C/C++ Ver 9</i> .	50 / 250
Npaths	Number of paths in module retrieved.	50.00 / 300.00
User	"(-1,0,1) = <b>Xcalltree.user</b> <i>N Lo Hi</i> " for all <i>N</i> = pair-number. The default supplied sample of <b>Xcalltree.user</b> chooses values at random.	10 / 100
Connections	Up and Down callers/callees from clicked function.	5 / 5



# Xdigraph Utility

This chapter explains the Xdigraph Utility, which is TCAT C/C++'s graphical utility for understanding a program's structure and flow. This chapter applies to all editions of TCAT C/C++.

---

## 7.1 Purpose

The **Xdigraph** utility draws digraphs based on archive files from *TCAT C/C++*. Digraphs are composed of **edges** and **nodes**. Edges are derived from segments (also known as logical branches) representing sets of consecutive program statements, or a program's "actions" (see Figure 33). Nodes are the places or "states" where the actions occur.

## 7.2 Xdigraph File Format

For information regarding the format of directed graph chart files, see Technical Appendix A.

### 7.3 Invoking Xdigraph

To invoke **Xdigraph** from the command line, type:

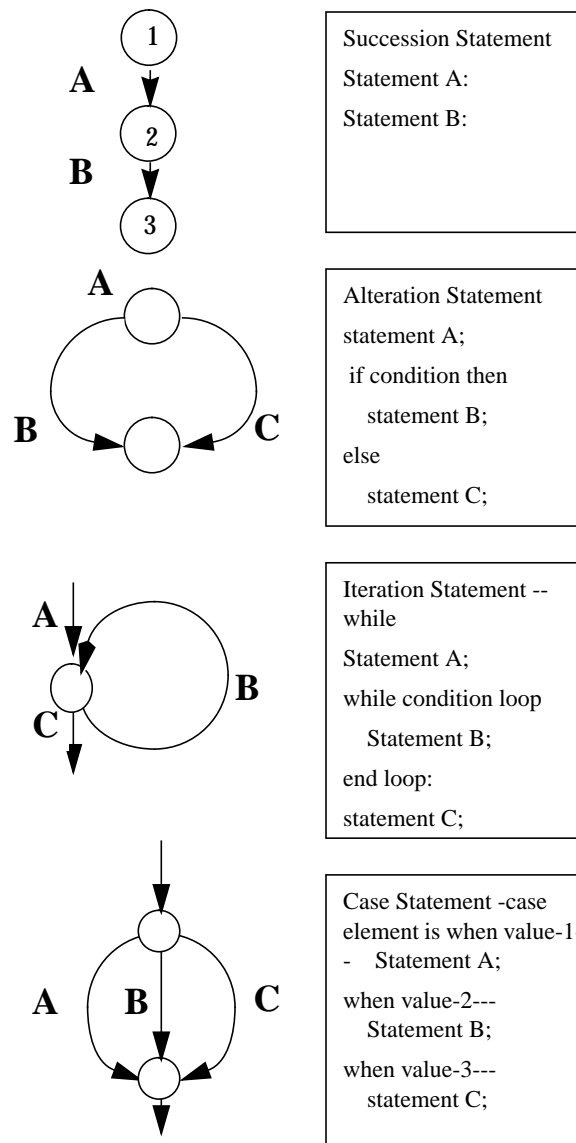
```
Xdigraph filename
```

This will result in a digraph being drawn onscreen based on the filename given. The available switches have the following values:

- |                 |                                                                                                                                                       |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| -A archive file | Archive file name. Default is 'Archive'.                                                                                                              |
| -B filename     | <i>Spine file</i> . This will change the text string for the digraph's nodes; the program will use the text settings for the filename typed after -B. |
| -H filename     | <i>Highlight specified file</i> . File called will come up in "highlight" mode; program searches for file with <i>.pth</i> extension.                 |
| -h              | This switch brings up <b>Xdigraph</b> 's help window.                                                                                                 |

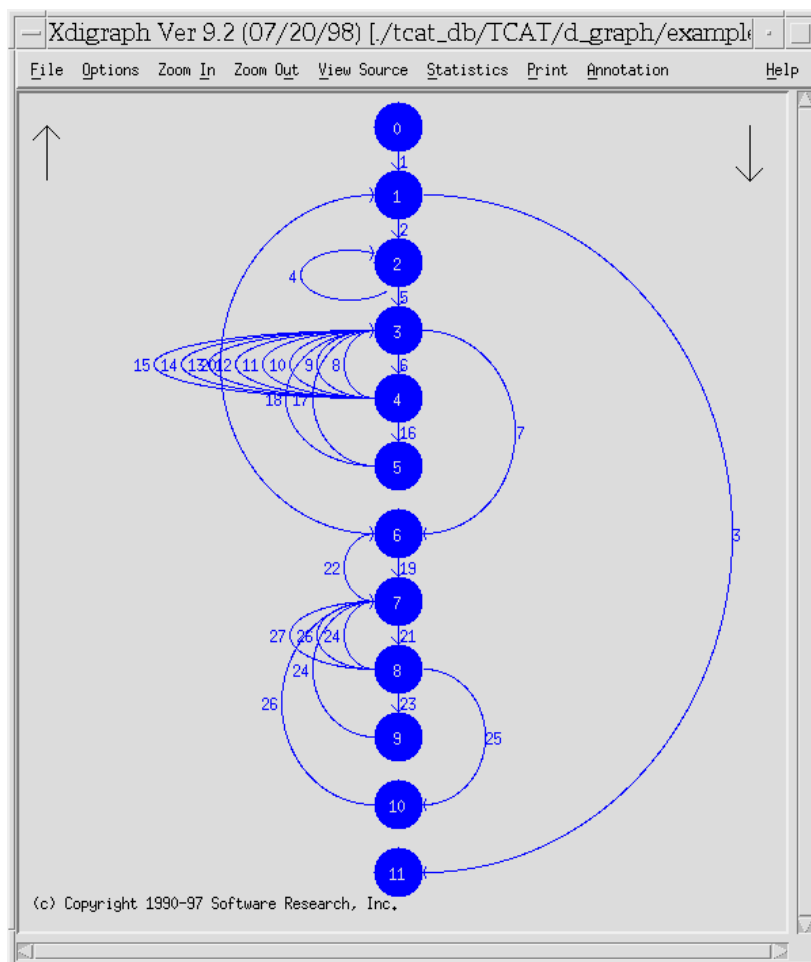
You can also invoke the utility without specifying a file by simply typing its name:

```
Xdigraph
```



**FIGURE 33** Program edges as represented in a digraph

## 7.4 Xdigraph Main Window



**FIGURE 34**

Xdigraph main window

Using **Xdigraph**, you can display a program's digraph and annotate it in a variety of ways. From **Xdigraph**'s Main Window menu bar, nine options are available.

**7.4.1 File**

This window allows you to select the file that will be displayed in the digraph.

**7.4.2 Options**

This window allows you to choose the characteristics of the nodes and edges displayed in the digraph, including shape, size, and color, as well as the scale for the **Zoom In** & **Zoom Out** options.

**7.4.3 Zoom In**

This option allows you to narrow the focus of the digraph, so that you can see it in more detail. There are maximum amounts that you can reduce or enlarge graphics, depending on what machine you are using.

**7.4.4 Zoom Out**

This option allows you to expand focus of the digraph, so that you can see it in wider perspective. There are maximum amounts that you can reduce or enlarge graphics, depending on what machine you are using.

**7.4.5 View Source**

This window allows you to view the source code for the current digraph.

**7.4.6 Statistics**

This window allows you to display pertinent statistics about the digraph, including node and edge counts, cyclomatic number, and path information.

**7.4.7 Print**

This window allows you to set the parameters and print the digraph.

**7.4.8      Annotation**

This window allows you to set the maximum and minimum thresholds for the nodes and edges in the digraph, as well as its path file.

**7.4.9      Help**

If you have a problem using **Xdigraph**, click on **Help**. Click your mouse on the **Action** pull-down menu and select **Search**. You will then get an **Enter String to search** dialog box. Click on the blank area and type the name of the option or function with which you need help.

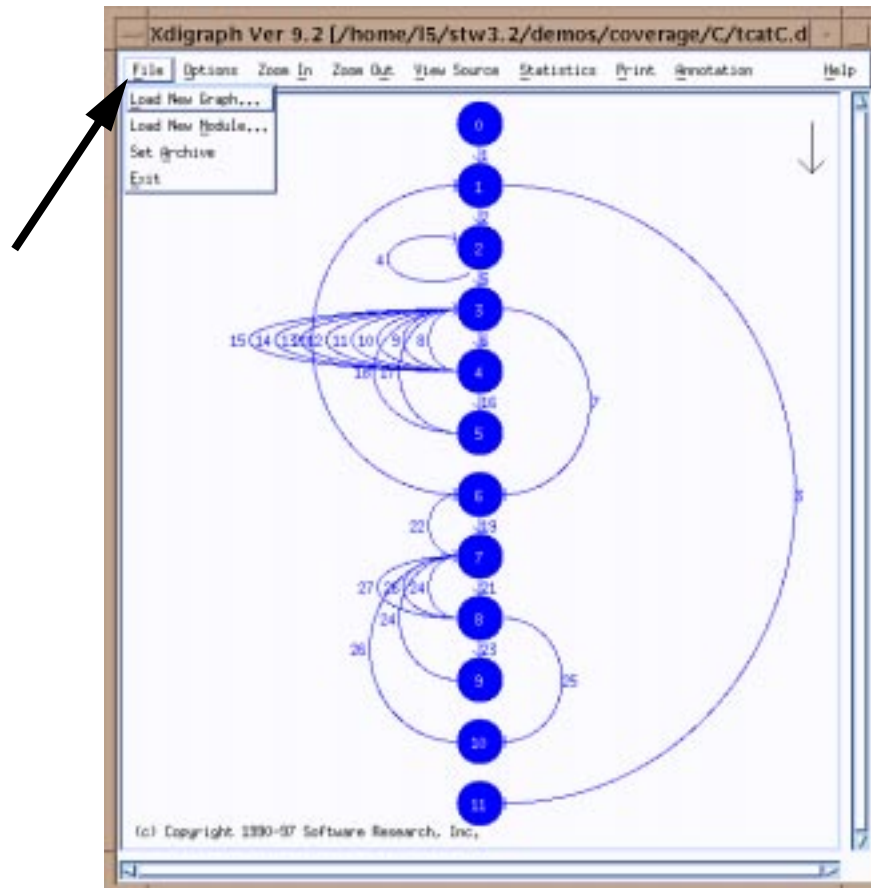
---

**NOTE:** These windows are explained in further detail on the following pages.

---



## 7.5 File Pull-Down Menu



**FIGURE 35** Digraph File Pull-Down Menu

### 7.5.1 Load New Graph

Click your mouse on the **File** pull-down menu. Drag the mouse to **Load New Graph**. The File Message Box Pop-Up (Figure 36 on page 97) will appear onscreen.

### 7.5.2 Load New Module

You use the **Load New Module** option if you have a multiple-digraph file and you want to choose a specific module in that file to be displayed.

When you click on this button the display shows you the set of available module names, taken from the multi-module digraph file that you have selected. You can then choose the module to be displayed. As soon as you click on **OK**, **Xdigraph** replaces the picture you have (if any) with the one corresponding to the named module.

If you don't have a multiple-module digraph file then this window may show no names. This is not an error but indicates that there are no module names specified.

### 7.5.3 Set Archive

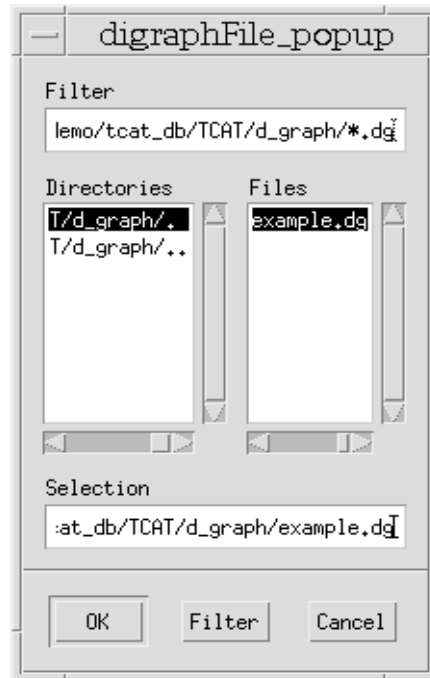
The default Archive file is "Archive" in the working directory, but you can change this to any file you wish using the "Set Archive" button. After you push the button you will be given a file-selection popup. Select the file you want to use as the Archive file and click on **Apply** to confirm that choice. The current name of the Archive file is shown in the filename section of the window. Default is "Archive" in the working directory.

### 7.5.4 Exit

To close the current digraph window, select **Exit** from this pull-down menu.

### 7.5.5 Digraph File Message Box

The message box in Figure 36 will pop up after you click the mouse on **Load New Graph** or **Load New Module**. The available options will allow you to choose the file to be represented in the digraph.



**FIGURE 36** Digraph File Message Box

**7.5.5.1 Filter**

Allows you to limit the number of files that will be searched for; only those ending in **.dg** will be included.

**7.5.5.2 Directories**

The directory from which the file is chosen to display in the digraph. Click on the chosen directory; it will show as darkened on the screen. Use the scroll bar at the bottom of this box if you cannot read the entire path-name of the directory.

**7.5.5.3 Files**

The actual file name selected to display in the digraph. Click on the file name, and the choice will be displayed in the **Selection** box. Use the scroll bar at the bottom of the box if you cannot read the entire filename.

**7.5.5.4 Selection**

Displays the file name selected in the **Files** box, or you can type in another name.

**7.5.5.5 OK**

Click **OK** when the desired file name is in the **Selection** box. The file named will then be represented as a digraph.

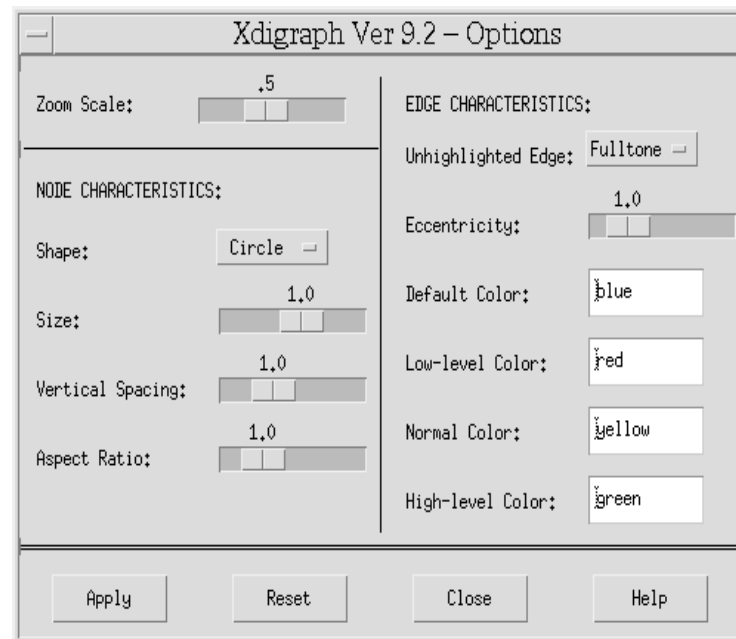
**7.5.5.6 Filter Button**

Clicking on this button will activate the filter limitations specified in the **Filter** box at the top of the window.

**7.5.5.7 Cancel**

To exit the window, without saving any changes, click on the **Cancel** button.

## 7.6 Options Window



**FIGURE 37** Xdigraph Options Window

This window allows you to choose the scale for the **Zoom In** and **Zoom Out** options, the size of the digraph's nodes, and the colors of its edges.

**7.6.1      Zoom Scale**

This setting affects the **Zoom In** and **Zoom Out** options. The default setting is 0.5, meaning a 50% reduction or enlargement in scale each time these buttons are used. To change the setting, move the slider left or right. Each 0.1 represents 10%, so if you slide the rule to .3, for example, the reduction and enlargements will be 30% each time. There are minimum and maximum amounts that you can reduce or enlarge graphics depending on what machine you are using.

## **7.6.2 Node Characteristics**

You can choose different sizes and shapes for the digraph's nodes. You can also change the space between nodes, and their height-to-width ratio, using this window.

### **7.6.2.1 Shape**

You have four choices for shapes: **Circle**, **Box**, **Oval** or **Outlined** (the circle is drawn but not filled). The default setting is **Oval**.

### **7.6.2.2 Size**

You can choose the size of the circle, box or oval. The default size is 1.0.

### **7.6.2.3 Vertical Spacing**

This is the amount of space between nodes. The default setting is 1.0.

### **7.6.2.4 Aspect ratio**

The height-to-width ratio (for ovals or box shapes only). The default setting is 1.4.

**7.6.3 Edge Characteristics**

**7.6.3.1 Unhighlighted Edge**

There are three choices: **Fulltone**, **Halftone** (dashes) or **Blank** (no visible lines). The default setting is **Fulltone**.

**7.6.3.2 Eccentricity**

Determines the curvature of the generated display. The value 1.0 means the edge between the two nodes is drawn as a semi-circle: bigger values make the picture wider, and smaller values narrower. The default setting is 0.6.

**7.6.3.3 Default Color**

Selects the basic color of the digraph's edges and nodes. The default setting is **blue**.

**7.6.3.4 Low-level Color**

In all cases, if the value of the chosen annotation is below the values indicated for Threshold 1, the display is done in the Low-level color. The default setting is **red**.

**7.6.3.5 Normal Color**

If the value of the chosen annotation is between Threshold 1 and Threshold 2, the Normal color is used. The default setting is **yellow**.

**7.6.3.6 High-level Color**

If the value of the chosen annotation is above the value stated in Threshold 2, then the High-level color is used. The default setting is **green**.

---

**NOTE:** If you have a monochrome display, then the three colors are expressed as a narrow, normal, and triple-wide line.

---



**7.6.3.7      Apply**

You must click on the **Apply** button for the current settings to take effect.

**7.6.3.8      Reset**

If you click on the **Reset** button, all the default settings will be restored to the **Options** window.

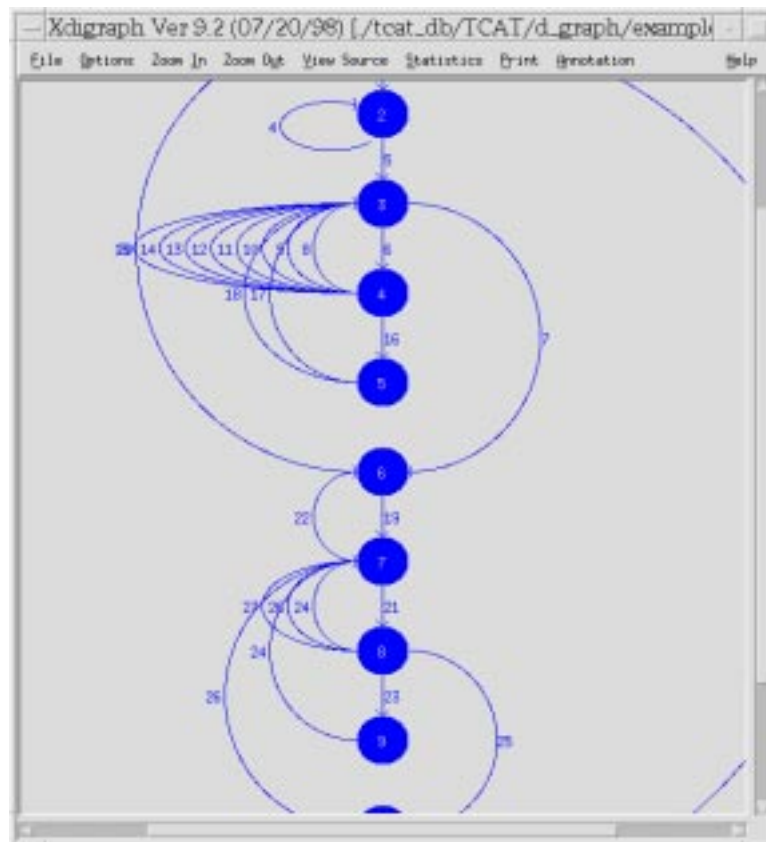
**7.6.3.9      Close**

If you click on the **Close** button, you will exit the **Options** window.

**7.6.3.10     Help**

If you have a problem using the **Options** window, click on **Help**. Click your mouse on the **Action** pull-down menu and select **Search**. You will then get an **Enter String to search** dialog box. Click on the blank area and type the name of the option or function with which you need help.

## 7.7 Zoom In/Zoom Out Window



**FIGURE 38** Zoom In feature illustrated

The zoom buttons allow for a narrower or wider perspective of the digraph, depending on what you require. Click on the **Zoom In** button once to narrow the focus of the digraph, and click on the **Zoom Out** button to get a wider perspective of the digraph. Notice the difference between the digraph in Figure 38, after clicking on **Zoom In** once, and the same digraph, depicted in Figure 34.

The arrow (triangle) symbols on the right-hand side and bottom of the window are scroll bars, which you can use to move vertically or horizontally while viewing the digraph.

**NOTE:** These features are limited by the display capabilities of your machine.

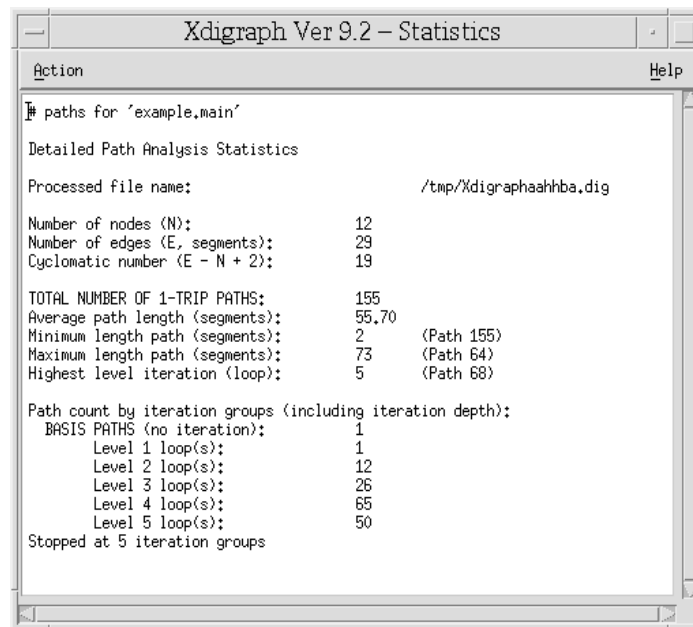
## 7.8 View Source Window



**FIGURE 39** View Source Option Window

This option displays the source code for the program depicted in the digraph. If you click on an edge segment number in the digraph's main window, the **View Source** display will move to and highlight that particular edge's source code. The source code for the edge selected will appear in the middle of the window.

## 7.9 Statistics Window



**FIGURE 40** Statistics Option Window

This window displays the relevant statistical information for the digraph. If the file you are processing has multiple digraphs on it, then only the displayed digraph is reflected in the **Statistics** calculation.

**WARNING:** In some cases, particularly if the digraph is very complex, the **Statistics** calculation will take a long time. Practical internal limits have been set on the STW facility that computes these statistics (i.e. `apg` from *TCAT-PATH*) but even so the calculation may show the “hour glass” waiting symbol.

When the limits are exceed you will see the error message that results in the display where the statistics would ordinarily reside.

The statistics generated in this window are always for the digraph that is on the display.

**7.9.1 File Name**

The name of the program studied in this particular digraph.

**7.9.2 Node and Edge Count**

The total number of nodes and edges in the digraph.

**7.9.3 Cyclomatic Number (Cyclomatic Complexity)**

A number which assesses program complexity according to the program's flow of control. This flow is based on the number and arrangement of decision statements in the code. The cyclomatic number can be calculated using the formula:

$$\text{cyclo} = e - n + 2$$

where **n** is the number of nodes in the graph, and **e** is the number of edges or lines connecting each node.

**7.9.4 Average, Minimum and Maximum Path Lengths**

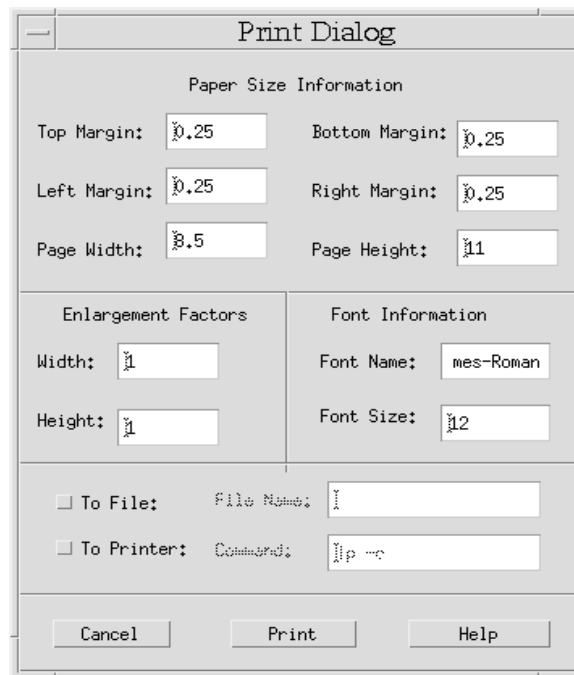
The mathematical mean of all the paths in the program, as well as (user-defined) minimum and maximum possible lengths.

**7.9.5 Path Count by Iteration Groups**

The path count by iteration groups is the total number of distinct equivalence classes of program flow, figured using the one-trip loop assumption (for details on how this computation is done, see the *TCAT-PATH* manual).

The total path count has been shown to be very highly correlated with the overall effort required to completely test a module.

## 7.10 Print Window



**FIGURE 41** Print Dialog Window

The image you see will be printed to a standard print device. This window will allow you to configure for your environment.

**7.10.1****Paper Size Information**

<b>Top Margin</b>	The distance from the top of the page to the first line. Default setting is 0.25 inches.
<b>Left Margin</b>	The distance from the left-hand side of the page to the first character of type. Default setting is 0.25 inches.
<b>Page Width</b>	The actual horizontal width of the paper you will be printing on. Default setting is 8.5 inches.
<b>Bottom Margin</b>	The distance from the bottom of the page to the last printed line. Default setting is 0.25 inches.
<b>Right Margin</b>	The distance from the right-hand side of the page to last character on the line. Default setting is 0.25 inches.
<b>Page Height</b>	Actual vertical length of the paper to be printed on. Default setting is 11 inches.

### 7.10.2      **Enlargement Factors**

The enlargement factors specify the size expansion, vertically or horizontally, to be applied to this particular print activity; in effect, the total number of 8.5 inch by 11.0 inch sheets onto which to draw the picture.

Selecting 1.0 means the picture will be printed on a single 8.5 inch x 11.0 inch sheet. Hence, width = 1.0 and height = 1.0 will fit the image on a standard page.

If you change the width to 2.0, for example, this means the picture will be drawn on two pages, i.e. in such a way that two 8.5 inch by 11.0 inch sheets can be pasted together to make a 17.0 inch by 11.0 inch image.

When more than one sheet is involved, the software numbers each page so that assembly into a larger diagram is simple and straightforward.

The software automatically sizes the image to fit into the smallest whole number of page equivalents. Also, the software sizes the diagram and the typefaces to "best fit" the specified size.

Some experimentation may be required to determine the optimum size for the diagram you are working with.

---

**NOTE:** The picture drawn by the printer always includes all of the information in the diagram, even if the entire diagram is not visible because of a zoom setting.

---



**7.10.3 Font Information**

The default font size, 12 point, and the default font name, Times-Roman, normally provide good quality pictures. Times-Roman at 12 point is commonly available on most printers.

You can choose different typesizes and type fonts depending on the sizes and fonts available on your computer.

**7.10.4 Print Locator**

**To File** Will create a PostScript (.ps) file, which you can use to have the digraph printed on any PostScript-compatible printer.

**To Printer** You must name the target printer where the print job will be sent.  
When the print job has been sent to either a .ps file or a printer, a message box, **Print action completed**, will pop up. Click **OK** to close it.

---

**NOTE:** The print option requires use of a PostScript-compatible printer. If your machine is not attached to a PostScript compatible printer, then the Print window option will be inoperative.

---

## 7.11 Annotation Window



---

**FIGURE 42** Annotation Thresholds Window

In many cases, annotation of the display is accomplished by showing the results of coverage testing, as reflected in the repository of multi-test coverage stored in the Archive file.

There are a number of ways to annotate the digraph. Typically this involves choosing a different color depending on where a particular parameter falls into user-specified ranges (thresholds).

There are five built-in annotation options and one user-defined annotation.

**7.11.1 Threshold 1 & 2**

Threshold 1 represents the lower limit, and Threshold 2 the upper limit desired for each annotation. The user can change the values of any threshold used by clicking in the window and typing in the new value. The values *won't* be applied to the current calltree unless you click the **Apply** button.

**7.11.2 None**

No annotation is shown.

**7.11.3 Nhits**

Number of times an edge is executed. The edge's color is based on this number. Default values: 1, 10

**7.11.4 N%**

The relative number of times an edge has been executed. The color depends on this number's relation to the highest number of times any edge is exercised. Default values: 10.00, 90.00.

**7.11.5 Nlines**

The number of code lines associated with the edge. Default values: 5, 25.

**7.11.6 User**

The outcome of calling a user-defined function, "Xdigraph.user", if that function exists, is the value used to color the display. Default values: 10, 100.

**7.11.7 Highlight**

The path highlight options permits you to see how a path set--typically one produced by **apg** (all paths generator)--applies to a particular digraph. If a path name is not specified, then it is automatically generated by **apg**.

Each path in the set is shown highlighted. The path number is always shown "on screen". You can move forward or backward in the path set using the mouse buttons as follows:

- Left button: move down one path in the path set (N-1)
- Middle button: quit the highlighting activity.
- Right button: Move up one path in the path set (N+1)

**7.11.8 Path File**

This indicates the file you have selected to represent in the digraph (optional — see note above).

**7.11.9 Apply**

If you click on the **Apply** button, all the settings changes made in the **Annotation Thresholds** window will be displayed on the digraph. You must click here to apply the changes.

**7.11.10 Reset**

If you click on the **Reset** button, all the default settings will be restored to the **Annotation Thresholds** window.

**7.11.11 Close**

If you click on the **Close** button, you will exit the **Annotation Thresholds** window.

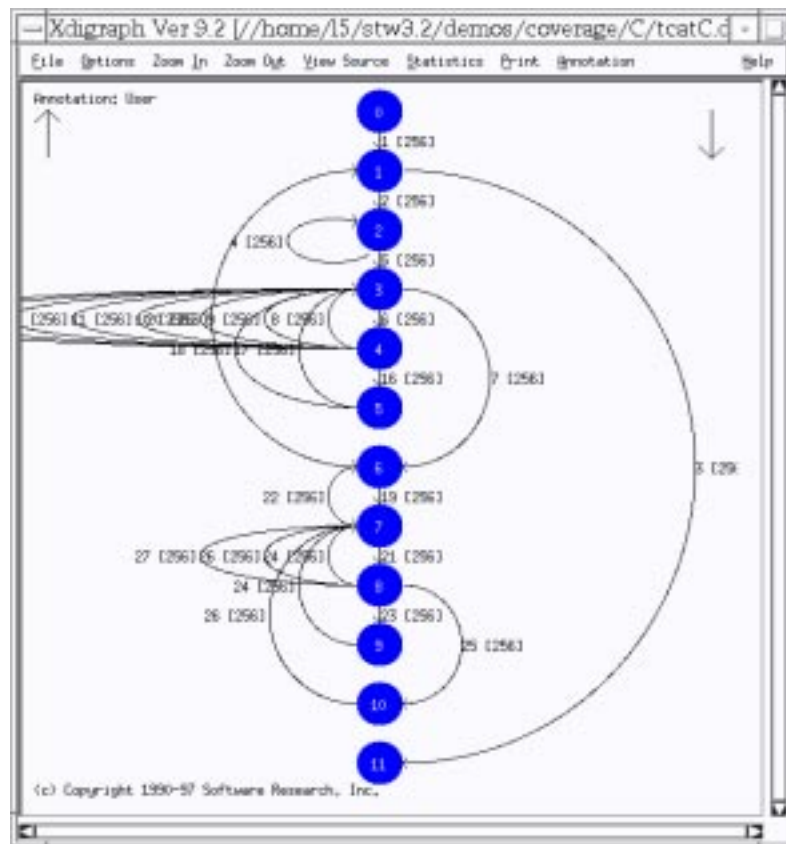
**7.11.12 Help**

If you have a problem using the **Annotation Thresholds** window, click on **Help**. Click your mouse on the **Action** pull-down menu and select **Search**. You will then get an **Enter String to search** dialog box. Click on the blank area and type the name of the option or function with which you are experiencing difficulty.

### 7.11.13 Colors

The colors of the digraph display are based on the annotation thresholds. They are selected in the Options window (see Section 7.6 for further details), and are used to distinguish the annotation to “low”, “normal”, and “high”. How these colors convey information is a function of which annotation is chosen.

**NOTE:** Whatever annotation option is selected for the digraph will be displayed in the upper left-hand corner of the main window, above an up-pointing arrow. In the example in Figure 43, the annotation is for **User**.



**FIGURE 43** Sample Annotation for **User** Threshold

## 7.12 Quick Reference Guide to Xdigraph Annotations

Function	Display Coloring Reflects What Information?	Preset Low/High
Nhits	Absolute number of hits per edge (segment), from local <b>Archive</b> file. The <b>Archive</b> file is must be <i>from TCAT C/C++</i> . If not, silence.	1 / 10
N%	Percent of total hits in module for this edge (segment), from local <b>Archive</b> file. The <b>Archive</b> file must be from <i>TCAT C/C++</i> . If not, silence.	10.00/90.00
Nlines	Number of source lines. The number of non-blank lines in the module is the metric used to color the display. Requires use of <i>TCAT C/C++ Ver 9</i> .	5/ 25
User	"-1,0,1) = <b>Xdigraph.user</b> <i>N Lo Hi</i> " for all <i>N</i> = edge-number. The default supplied sample script does something naive.	10 / 100
Highlight	Highlights <i>N</i> th path, beginning at <i>N=1</i> . (Path File) Left button moves up one path; right button moves down one path. If no path file is specified, <b>apg</b> will generate one.	N/A

TABLE 3

Quick Reference Guide for Xdigraph Annotations

# C/C++ Instrumentor Engine

This appendix provides a guide to TCAT C/C++ Version 9's new integrated C and C++ instrumentor. This appendix applies to all editions of TCAT C/C++.

---

## A.1 Instrumentor Description

*TCAT C/C++* instruments the source code of the system to be tested by inserting function calls at each logical branch and call pair. The instrumentation does not affect the functionality of the program. When compiled, linked and executed, the instrumented program will behave normally, but writes coverage data to a trace file. There is some performance overhead related to the data collection process, but the overhead varies with the choice of the runtime used. The trace files are processed by several kinds of report generators.

There are two versions of the instrumentor engine, one for C programs (**ic**) and one for C++ programs (**icpp**).

**A.1.1 Files Generated**

In operation, *TCAT C/C++* instrumentor parses candidate source code, looking for logical branches and/or call pairs, and generates auxiliary files that are used by other parts of the system. The following are files that *TCAT C/C++* uses and produces.

<b><i>tcat.rc</i></b>	<b><i>TCAT Control File.</i></b> This file controls certain features of the instrumentation process and also is used to specify options to the runtime process.
<b><i>filename.i</i></b>	Instrumented Source File. <b><i>file.i</i></b> is produced from <b><i>file.c</i></b>
<b><i>tcat_db*</i></b>	<b><i>TCAT</i></b> Database Directory. During operation, <b><i>TCAT</i></b> creates the <i>tcat_db</i> directory if it does not already exist. This database directory contains files as follows:
<b><i>Project File</i></b>	This file contains a list of fully expanded path names that specify where the instrumentor is to take its source.
<b><i>Digraph File</i></b>	This file contains the complete digraph information for all modules processed.
<b><i>Calltree File</i></b>	This file contains the complete calltree information for all modules processed.
<b><i>Inheritance File</i></b>	This file contains the complete inheritance tree information for all methods processed (applies to C++ only).
<b><i>Module Definition File</i></b>	<p>This file contains complete module definition information, in a format compatible with a trace-file's or archive file's type -n record. This information is used in all of the source representation processes supported by <b><i>TCAT</i></b>.</p> <p>There is also a C version of this same information, set up as a C structure format so that it can be used in cross-testing and embedded applications.</p>



### A.1.2 Command Line Invocation Summary

The syntax for command line invocation of either **ic** or **icpp** is as follows:

```
ic file.ext
[-TCAT-A]
[-TCAT-Cmd driver]
[-TCAT-C1]
[-TCAT-DI]
[-TCAT-E]
[-TCAT-FN]
[-TCAT-FULL]
[-TCAT-G]
[-TCAT-H]
[-TCAT-I]
[-TCAT-K]
[-TCAT-O file_name]
[-TCAT-OF .suffix]
[-TCAT-PD name]
[-TCAT-PN name]
[-TCAT-REL]
[-TCAT-S0]
[-TCAT-S1]
[-TCAT-SI]
[-TCAT-WC]
[-TCAT-WN]
[-Ddefs[=val]]
[-Ipath]
[-Uundefs]
```

These commands instrument submitted C language file(s).

The following instrumentor switches may be used to vary the processing and reports generated by the instrumentor. The instrumentor switches are listed in alphabetical order.

Note that the commands are prefixed with **-TCAT**. This is done because all other switches are passed to the underlying C or C++ compiler. The prefix indicates that these switches are for TCAT C/C++ processing.

file.ext	Instrumented File Specification(s). File(s) to be instrumented. The extension can be <b>c</b> or <b>i</b> or <b>cc</b> (for C++).
	If there are multiple files, each one is processed in the order presented, and they are treated as if they have been concatenated together.
-TCAT-A	ANSI Recognition Switch. If present, the instrumentor recognizes only the ANSI version of C or C++. -TCAT-Cmd driver Compiler Driver Command Switch. Default driver is <b>cc</b> .

-TCAT-C1	C1 Instrumentation Switch. If this switch is present, then the instrumentor inserts a function call in each segment, or logical branch. This is the preset default.
-TCAT-DI	De-Instrumented File Switch. This switch turns off instrumentation for objects specified in the file <i>tcat_db/projectname/projectname.di</i> . Note that specifying objects in this file without using this switch will not turn off instrumentation.
-TCAT-E	Print Error Messages Switch. Enables sending error messages to standard output. If not present, then error messages are suppressed.
-TCAT-FN	This switch turns off call-pair instrumentation for call-pair lists specified in the file: <i>tcat_db/projectname/projectname.di</i> . Note that specifying call-pair lists in this file without using this switch will not turn off instrumentation
-TCAT-FULL	Store tcat data using full path information.
-TCAT-G	Instrumented File Disposition Switch. Normally the instrumentor does not keep the instrumented file, it already having been used to produce the instrumented output. When this switch is present the instrumented files are retained.
-TCAT-H	Help Message Switch. Prints out the set of valid switches.
-TCAT-I	Instrumentation only, Do not invoke the compiler.
-TCAT-K	K&R C Recognition Switch. If present, the instrumentor recognizes K&R C.
-TCAT-O file	Output File Specification. The output of the instrumentation process is directed to the named file (default is file.i).
-TCAT-OF .suffix	Output Suffix Specification. Output of instrumentation process is directed to file.suffix instead of file.i.
-TCAT-PD name	Specified tcat project directory.
-TCAT-PN name	Specified tcat project name.

-TCAT-REL	Store tcata data using relative path information.
-TCAT-S0	S0 Instrumentation Switch. If this switch is present, then the instrumentor inserts a function call in each module. This will tell you which functions are actually called during the invocation of the program, but it does not indicate the callee functions. To do this, you need to use the -S1 switch.
-TCAT-S1	S1 Instrumentation Switch. If this switch is present, then the instrumentor inserts a function call in each call pair.
-TCAT-I	Only instrument items listed in .si file.
-TCAT-WC	Causes the instrumentor not to define macro _WCHAR_T.
-TCAT-WN	Causes the instrumentor to define macro _WCHAR_T. This is the default.
-Ddefs[=val]	Establish Definition Switch. Establishes a definition that is passed on to the compiler.
-Ipath	Include File Search Path Specification. Specifies the path on which to resolve the search for <b>#include</b> files.
-Uunefs	De-Establish (Undefine) Definition Switch. Removes a definition that is passed on to the compiler.

**A.1.3 Instrumentation Function Names**

The instrumentation process involves inserting function names into the source program. The function names for TCAT-instrumented programs are:

<b>SegHit()</b> ;	For entry segment, switch segments.
<b>CprHit()</b> ;	For S1 coverage of call pairs.
<b>ExpHit()</b> ;	For C1 coverage <b>if</b> 's, <b>while</b> 's and <b>for</b> 's.
<b>Strace()</b> ;	Start trace operations (this is an optional call).
<b>Ftrace()</b> ;	Finish trace operations, flush buffer, and close tracefile.

#### A.1.4 Instrumentor Inline Directives

It is possible to control instrumentation from within the processed “C” or “C++” file, using the following instrumentor directives to turn off/on all instrumentation (but keep the segments and call pairs numbered correctly):

```
/* TCAT OFF */  
/* TCAT ON */
```

#### A.1.5 Instrumentation Database Definitions

This section outlines the files that are used in the instrumentation database stored in the *tcat\_db* directory. This information is used throughout the TCAT C/C++ system.

#### A.1.6 Environment Variables

There are two environment variables that govern how *TCAT C/C++* operates.

1. *\$TCAT\_PROJECT\_DIR*, with default “.”, is the name of the project directory.
2. *\$TCAT\_PROJECT\_NAME*, with default “TCAT”, is the name of the project.

When *ic* or *icpp* executes, it stores results in these files and directories:

```
$TCAT_PROJECT_DIR/tcat_db/$TCAT_PROJECT_NAME/$TCAT-  
PROJECT_NAME.mdf  
$TCAT_PROJECT_DIR/tcat_db/$TCAT_PROJECT_NAME/d_graph/  
$TCAT_PROJECT_DIR/tcat_db/$TCAT_PROJECT_NAME/c_graph/
```

**A.1.7 d\_graph Files**

The digraphs for each function are put into files which are named with the same basename as the file from which they originated, with any file-name suffix stripped off.

The format of each *d\_graph* file is a set of blank delimited (white space delimited) lines composed as follows:

```
tail head edge fun_id type filename
lbeg lend byte_beg byte_end string
result [byte1 byte2]
```

where the fields have the following meanings:

tail	The tail node number (string).
head	The head node number (string).
edge	The TCAT C/C++ assigned edge number (string).
fun_id	The number of the function, whose name is found in the <b>mdf</b> file.
fun_id	The type of statement which gave rise to the edge.
filename	The filename where the original text of the program was found.
lbeg	The beginning line number, in the named file, where the tail node is found.
lend	The ending line number, in the named file, where the head node is found.
byte_beg	The beginning byte number, in the named file, where the tail node is found.
byte_end	The ending byte number, in the named file, where the head node is found.
string	The text string associated with the logical expression that headed the segment.
result	The result corresponding to this edge, e.g. <b>T</b> or <b>F</b> or <b>36</b> (for switch outcome).
[byte1 byte2 ]	Currently "0 0"; reserved for expansion.

A sample *d\_graph* file is listed in Section A.1.10.1

**A.1.8 c\_graph Files**

The call-trees for each processed file are put into files which are named with the same basename as the file from which they originated, with any filename suffix stripped off.

The format of each **c\_graph** file is as a set of blank delimited (white space delimited) lines composed as follows:

```
file.caller callee callpair_id module_id
source_file line 0 0 Segment_id
```

where the fields have the following meanings:

file.caller	The file name (given as a prefix up to the right-most "." in the token, and the name of the calling function (the "caller").
file.callee	The name of the called function.
callpair_id	The assigned number of the call pair.
module_id	The assigned number of the module. This number points into the <b>mdf</b> file.
source_file	The name of the source file that gave rise to the call pair.
line	The line number of the source file where the call pair exists.
0 0	These two fields are pre-set to be "0 0".
Segment_id	(Reserved for future releases).

**A.1.9 Module Definition Files (mdf, mdf.c)**

The mdf file contains basic information about the location of text fragments for every segment and every call pair in all processed files.

The **mdf** file has the following format (fields in [ ]'s are future):

```
project-name #segs #CPs [#rels]
file.name.function_id type #segs #CPs [#rels]
file.name.function_id type #segs #CPs [#rels]
file.name.function_id type #segs #CPs [#rels]
...
```

where the first line identifies:

<b><i>project-name</i></b>	This is the name of the “project” from which the data is taken.
<b><i>#segs</i></b>	This is the total number of segments in the project.
<b><i>#CPs</i></b>	This is the total number of call-pairs in the project.

The subsequent lines' fields have the following meanings:

<b><i>file.name</i></b>	This token contains, first, the name of the file in which the function name was found, and second, after the right-most “.”, the name of the function.
<b><i>function_id</i></b>	This is the unique numeric identifier for that function, as found in the filename which prefixes the function name.
<b><i>type</i></b>	This is the type of function that was processed, according to the key: 84 = static function; 111 = member function. Note: These numbers are implementation specific. Additional function types and different codes will be added in the future. At present this function type information is not used.
<b><i>#segs</i></b>	The number of segments in the function.
<b><i>#CPs</i></b>	The number of call pairs in the function.



## A.1.10 Example Instrumentation Database Files

Below, examples of the database files are provided.

### A.1.10.1 **d\_graph** File

This is the front part of a typical **d\_graph** file:

```

0 1 0 232 0 cfvtable.cc 27 0 0 0 (1) 0 0 0
0 1 0 233 0 cfvtable.cc 41 0 0 0 (1) 0 0 0
1 2 1 233 1 cfvtable.cc 41 0 0 0 (vtype==0) 1 0 0
2 3 3 233 1 cfvtable.cc 43 0 0 0 (!(mptr!=0)) 1 0 0
2 3 4 233 1 cfvtable.cc 43 0 0 0 (!(mptr!=0)) 0 0 0
3 4 5 233 1 cfvtable.cc 49 0 0 0 (!(pure_err_func!=0)) 1 0 0
3 4 6 233 1 cfvtable.cc 49 0 0 0 (!(pure_err_func!=0)) 0 0 0
1 4 2 233 1 cfvtable.cc 51 0 0 0 (vtype==0) 0 0 0
0 1 0 234 0 cfvtable.cc 60 0 0 0 (1) 0 0 0
0 1 0 235 0 cfvtable.cc 72 0 0 0 (1) 0 0 0
1 2 1 235 1 cfvtable.cc 92 0 0 0 (ftype->typ!=type::fp_ptr_t) 1 0 0
1 2 2 235 1 cfvtable.cc 97 0 0 0 (ftype->typ!=type::fp_ptr_t) 0 0 0
0 1 0 236 0 cfvtable.cc 134 0 0 0 (1) 0 0 0
1 2 1 236 1 cfvtable.cc 171 0 0 0 (can_be_virt) 1 0 0
1 2 2 236 1 cfvtable.cc 202 0 0 0 (can_be_virt) 0 0 0
2 3 3 236 1 cfvtable.cc 211 0 0 0 (can_be_virt) 1 0 0
2 3 4 236 1 cfvtable.cc 222 0 0 0 (can_be_virt) 0 0 0
0 1 0 237 0 cfvtable.cc 235 0 0 0 (1) 0 0 0
0 1 0 238 0 cfvtable.cc 253 0 0 0 (1) 0 0 0
0 1 0 239 0 cfvtable.cc 282 0 0 0 (1) 0 0 0
0 1 0 240 0 cfvtable.cc 295 0 0 0 (1) 0 0 0
1 2 1 240 1 cfvtable.cc 296 0 0 0 (e->m.func==0) 1 0 0
1 2 2 240 1 cfvtable.cc 300 0 0 0 (e->m.func==0) 0 0 0
2 3 3 240 1 cfvtable.cc 303 0 0 0 (e->m.func->f.virtual_index==0) 1 0 0
2 3 4 240 1 cfvtable.cc 309 0 0 0 (e->m.func->f.virtual_index==0) 0 0 0
0 1 0 241 0 cfvtable.cc 327 0 0 0 (1) 0 0 0
1 2 1 241 5 cfvtable.cc 332 0 0 0 (kind) 2756 0 0
1 2 2 241 5 cfvtable.cc 335 0 0 0 (kind) 61824 0 0
1 2 3 241 5 cfvtable.cc 338 0 0 0 (kind) 61824 0 0
2 3 4 241 1 cfvtable.cc 346 0 0 0 (kind!=external) 1 0 0
2 3 5 241 1 cfvtable.cc 350 0 0 0 (kind!=external) 0 0 0
0 1 0 242 0 cfvtable.cc 359 0 0 0 (1) 0 0 0
0 1 0 243 0 cfvtable.cc 368 0 0 0 (1) 0 0 0
0 1 0 244 0 cfvtable.cc 389 0 0 0 (1) 0 0 0
0 1 0 245 0 cfvtable.cc 396 0 0 0 (1) 0 0 0
1 2 1 245 1 cfvtable.cc 396 0 0 0 (vkind!=external) 1 0 0
1 2 2 245 1 cfvtable.cc 400 0 0 0 (vkind!=external) 0 0 0
0 1 0 246 0 cfvtable.cc 411 0 0 0 (1) 0 0 0
0 1 0 247 0 cfvtable.cc 417 0 0 0 (1) 0 0 0

```

**A.1.10.2      c\_graph File**

This is the front part of a typical *c\_graph* file:

```
example.main printf 1 0 example.c 46 0 0 2
example.main printf 2 0 example.c 48 0 0 4
example.main gets 3 0 example.c 50 0 0 5
example.main printf 4 0 example.c 51 0 0 5
example.main proc_input 5 0 example.c 52 0 0 5
example.main printf 6 0 example.c 55 0 0 8
example.main printf 7 0 example.c 58 0 0 9
example.main printf 8 0 example.c 61 0 0 10
example.main printf 9 0 example.c 64 0 0 11
example.main printf 10 0 example.c 67 0 0 12
example.main printf 11 0 example.c 70 0 0 13
example.main printf 12 0 example.c 73 0 0 14
example.main printf 13 0 example.c 76 0 0 15
example.main printf 14 0 example.c 80 0 0 17
example.main printf 15 0 example.c 85 0 0 19
example.main _filbuf 16 0 example.c 86 0 0 19
example.proc_input strlen 1 1 example.c 117 0 29 3
example.proc_input chk_char 2 1 example.c 135 0 0 16
example.proc_input strlen 3 1 example.c 141 0 0 20
example.proc_input chk_char 4 1 example.c 142 0 0 21
example.proc_input printf 5 1 example.c 149 0 0 22
```

```
-----
Caller Callee Callpair_id Module_id source_file line 0 0
Segment_id:
```

### A.1.10.3 mdf File

This is the front part of a typical mdf file:

```
TCAT 1904 1903
analc.declare_ident(sym*,tokentype,name_kinds,int&) 0 84 45 11
analc.transfer_defaults(void,type*) 1 84 11 2
analc.distinct(int,type*) 2 84 15 7
analc.all_but_return(int,type*) 3 84 1 1
analc.check_overload(sym*,type*,cpp_ext_linkage,sym*,int&) 4 84 21 11
analc.new_func_sym(void,type*,int,tokentype,cpp_ext_linkage) 5 111 13 5
analc.check_copy_constructor(void,type*) 6 84 13 7
analc.check_op_overload(void,type*) 7 111 100 21
analc.func_checks(void,sym*&,sym*,tokentype,cpp_ext_linkage,int&) 8 111 37 17
analc.check_throws(void,int,type_list*) 9 84 15 8
analc.func_declare(sym*,spelling,tokentype,int,cpp_ext_linkage,int,type_list*,int&) 10 111
15 8
analc.old_arg_types(void) 11 84 39 23
analc.set_qualifier(spelling,type*,spelling,int) 12 84 13 8
analc.find_qfunc(sym*,type*,spelling,int,type_list*) 13 111 19 12
analc.func_define(void,base_debug_position,type*,spelling,token-
type,int,int,int,type_list*) 14 111 35 39
analc.type_define(void,int,type*,spelling,Cbase_debug_position&) 15 111 43 19
analc.var_declare(sym*,spelling,tokentype,int,Cbase_debug_position&) 16 111
37 15
analc.stat_mem_def(sym*,type*,spelling,tokentype,int,Cbase_debug_position&) 17 84 15 10
analc.static_inits(void,type*,sym*) 18 111 13 16
analc.file_decl(void) 19 84 111 89
analc.parse_declarations(void) 20 111 27 35
analc.analys(void) 21 111 9 8
asmdecl.asm_decl(void) 22 111 3 6
asminitializer.asm_initializer::fill(void,long) 23 111 5 4
asminitializer.asm_initializer::bf_data(void,Ulong,int,int) 24 111 1 0
asminitializer.asm_initializer::inc_findex(Uint) 25 111 5 0
asminitializer.asm_initializer::start_var(void,sym*) 26 111 3 9
asminitializer.asm_initializer::start_array(void,type*) 27 111 1 2
asminitializer.asm_initializer::end_array(void,type*) 28 111 1 1
asminitializer.asm_initializer::start_struct(void,type*) 29 111 1 1
asminitializer.asm_initializer::end_struct(void,type*) 30 111 1 1
asminitializer.asm_initializer::advance_elt(void) 31 111 3 3
asminitializer.asm_initializer::start_field(void,sym*) 32 111 3 3
```



# Resource File Variables

This appendix describes the resource files supplied with TCAT C/C++, and applies to all editions of the product.

---

## B.1 Overview

Within the **TCAT C/C++** graphical user interface (GUI), command line switches are used to affect the behavior of the application. The switches determine such information as which compiler is used and what flags must be specified for the instrumentor. These switches vary from platform to platform, though their meaning is consistent.

The resource files provided (and documented in this appendix) assume that the standard compilers supplied with each platform are used. If a non-standard compiler is used, you will have to change this resource file. Since machine and compiler configurations vary, you may have different requirements and results, even if the standard compiler is used.

To load these resource files, use the **Load Settings** option in the **TCAT C/C++** GUI, and select the \*.res file. If these settings are incorrect, determine the exact switch settings for your configuration. Save your settings using the **Save Setting** option.

The next section discusses the meaning of each variable, and gives the example resource file settings for each platform.

## B.2 Variable Meaning

The variables within the resource file have the following meanings:

SR\*config.option.instrumentS0: The state of the S0 instrumentation button within the GUI.

SR\*config.option.runTarget: The name of the file output from linking.

SR\*config.option.linkFlags: The flags used when the Link option is selected.

SR\*config.option.instrumentC++: The state of the C++ language button within the GUI.

SR\*config.option.buildCommand: The command used when the Build option is selected.

SR\*config.option.linkCommand: The command used when the Link option is selected.

SR\*config.option.instrumentAnsiC: The state of the ANSI C language button within the GUI.

SR\*config.option.instrumentKRC: The state of the K&R C language button within the GUI.

SR\*config.option.runArgs: The command line arguments used when the Run option is selected.

SR\*config.option.linkRuntimeModule: The name of the runtime module used when the Link option is selected.

SR\*config.option.linkLibraryFlags: The library flags used when the Link option is selected.

SR\*config.option.compilerCommand: The name of the compiler.

SR\*config.option.buildFlags: The flags used when the Build option is selected.

SR\*config.option.instrumentC1: The state of the C1 instrumentation button within the GUI.

SR\*config.option.instruCommand: The name of the instrumentor, ic9 for "C" applications, icpp9 for "C++" applications.

SR\*config.option.instrumentS1: The state of the S1 instrumentation button within the GUI.

SR\*config.option.compilerFlags: The flags used when the compile option is selected.

The most important of these resources is the `SR*config.option.compiler-Flags`, as this setting tells the instrumentor what flags are “assumed” by the compiler. Each compiler assumes that the `-I` and `-D` switches are set to certain values, based on the machine it is installed on and other specific information. In order for the preprocessing to be successful, the instrumentor must assume the same switches. This resource indicates the pertinent information.

The compiler flags resource uses the correct flags for the standard compilers for each platform, but they might have to be changed for your particular configuration, compiler or machine.

Two platforms must have the `-tcatt-of` flag set for this variable when instrumenting “C++”. This switch indicates that the instrumentor use a alternate file extension. The standard extension, `*.i`, cannot be used with DEC Alpha or SGI platforms.

For the DEC Alpha, the switch must specify that the instrumentor use the `*.ixx` extension.

For the SGI, the switch must indicate that the instrumentor uses one of the following extensions: `*.C`, `*.c`, `*.c++`, `*.cc` or `*.cxx`.

## **B.3 Platform-Specific Examples**

Below are the resource files for each supported platform. Each section contains the resource file for “C” and “C++”.

### **B.3.1 DEC Alpha**

This is the resource file for “C” on the DEC Alpha platform.

```
! Flags needed to instrument K&R C code on the DECalpha
! for compilation with cc
```

```
SR*config.option.instrumentS0:False
SR*config.option.runTarget:a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:False
SR*config.option.buildCommand:make -f
SR*config.option.linkCommand:cc -o
SR*config.option.instrumentAnsiC:False
SR*config.option.instrumentKRC:True
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule:crun4.o
SR*config.option.linkLibraryFlags:
SR*config.option.compilerCommand:cc
SR*config.option.buildFlags:
SR*config.option.instrumentC1:True
SR*config.option.instruCommand:ic9
SR*config.option.instrumentS1:True
SR*config.option.compilerFlags:-c -D__alpha
-D__host_alpha -D__osf__
```

This is the resource file for “C++” on the DEC Alpha platform.

```
! Flags needed to instrument C++ code on the DECalpha
! for compilation with cxx
```

```
SR*config.option.instrumentS0:False
SR*config.option.runTarget:a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:True
SR*config.option.buildCommand:make -f
SR*config.option.linkCommand:cxx -o
SR*config.option.instrumentAnsiC:False
SR*config.option.instrumentKRC:False
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule:crun4.o
SR*config.option.linkLibraryFlags:
```



```
SR*config.option.compilerCommand:cxx
SR*config.option.buildFlags:
SR*config.option.instrumentC1:True
SR*config.option.instruCommand:icpp9
SR*config.option.instrumentS1:True
SR*config.option.compilerFlags:-c -tcat-of .ixx
    -D__alpha -D__DECCXX -D__DECCXX_VER=50090003
    -D__host_alpha -D__osf__ -D_ANSI_C_SOURCE -I/
    usr/include/cxx
```

**B.3.2 Silicon Graphics, Inc.**

This is the resource file for “C” on the SGI platform.

```
! Flags needed to instrument ANSI C code on the SGI
! for compilation with cc

SR*config.option.instrumentS0:      False
SR*config.option.runTarget:         a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:     False
SR*config.option.buildCommand:      make -f
SR*config.option.linkCommand:       cc -o
SR*config.option.instrumentAnsiC:    True
SR*config.option.instrumentKRC:      False
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule: crun4.o
SR*config.option.linkLibraryFlags:
SR*config.option.compilerCommand:   cc
SR*config.option.buildFlags:
SR*config.option.instrumentC1:       True
SR*config.option.instruCommand:      ic9
SR*config.option.instrumentS1:       True
SR*config.option.compilerFlags:     -c -
    D__EXTENSIONS__ -D__INLINE_INTRINSICS -Dsgi -D__sgi -
    Dhost_mips -D__unix -D__host_mips -D_SVR4_SOURCE -
    D_MODERN_C -D_SGI_SOURCE -D__DSO__ -DSYSTYPE_SVR4 -
    D_SYSTYPE_SVR4 -D_LONGLONG -D__mips -D_MIPSEB -D_CFE
    -D_MIPS_FPSET=16 -D_MIPS_ISA=_MIPS_ISA_MIPS1
    -D_MIPS_SIM=_MIPS_SIM_ABI32 -D_MIPS_SZINT=32
    -D_MIPS_SZPTR=32 -D_MIPS_SZLONG=32
```

This is the resource file for “C++” on the SGI platform.

```
! Flags needed to instrument C++ code on the SGI
! for compilation with CC
```

```
SR*config.option.instrumentS0:      False
SR*config.option.runTarget:         a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:     True
SR*config.option.buildCommand:      make -f
SR*config.option.linkCommand:       CC -o
SR*config.option.instrumentAnsiC:    False
SR*config.option.instrumentKRC:      False
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule: crun4.o
SR*config.option.linkLibraryFlags:
SR*config.option.compilerCommand:   CC
SR*config.option.buildFlags:
SR*config.option.instrumentC1:       True
SR*config.option.instruCommand:      icpp9
SR*config.option.instruments1:       True
SR*config.option.compilerFlags:     -c -
    D__EXTENSIONS__ -D__INLINE_INTRINSICS -Dsgi -
    D__sgi -Dhost_mips -D__unix -D__host_mips -
    D_SVR4_SOURCE -D_MODERN_C -D_SGI_SOURCE -D__DSO__
    -DSYSTYPE_SVR4 -D_SYSTYPE_SVR4 -D_LONGLONG -
    D__mips -D_MIPSEB -D_CFE -D_MIPS_FPSET=16 -
    D_MIPS_ISA=_MIPS_ISA_MIPS1
    -D_MIPS_SIM=_MIPS_SIM_ABI32 -D_MIPS_SZINT=32
    -D_MIPS_SZPTR=32 -D_MIPS_SZLONG=32 -I/usr/include/
    CC
```

**B.3.3 Solaris**

This is the resource file for “C” on the Solaris platform.

```
! Flags needed to instrument K&R C code under Solaris
! for compilation with cc
```

```
SR*config.option.runTarget:          a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:       False
SR*config.option.buildCommand:        make -f
SR*config.option.instrumentAnsiC:      False
SR*config.option.instrumentKRC:        False
SR*config.option.compilerCommand:     cc
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule:   crun4.o
SR*config.option.linkLibraryFlags:    -lm
SR*config.option.buildFlags:
SR*config.option.linkCommand:         cc -o
SR*config.option.instrumentC1:         True
SR*config.option.instruCommand:        ic9
SR*config.option.compilerFlags:       -c
SR*config.option.instrumentS1:         True
SR*config.option.instrumentS0:         False
```

This is the resource file for “C++” on the Solaris platform.

```
! Flags needed to instrument C++ code under Solaris
! for compilation with CC

SR*config.option.runTarget:           a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:       True
SR*config.option.buildCommand:        make -f
SR*config.option.instrumentAnsiC:      False
SR*config.option.instrumentKRC:        False
SR*config.option.compilerCommand:     CC
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule:    crun4.o
SR*config.option.linkLibraryFlags:    -lm
SR*config.option.buildFlags:
SR*config.option.linkCommand:          CC -o
SR*config.option.instrumentC1:         True
SR*config.option.instruCommand:        icpp9
SR*config.option.compilerFlags:        -c -I/opt/
    SUNWspro/SC2.0.1/include/cc
SR*config.option.instrumentS1:         True
SR*config.option.instrumentS0:         False
```

**B.3.4 Sun**

This is the resource file for “C” on the Sun platform.

```
! Flags needed to instrument K&R C code under SunOS
! for compilation with cc
```

```
SR*config.option.runTarget:          a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:       False
SR*config.option.buildCommand:        make -f
SR*config.option.instrumentAnsiC:      False
SR*config.option.instrumentKRC:        False
SR*config.option.compilerCommand:     cc
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule:   crun4.o
SR*config.option.linkLibraryFlags:    -lm
SR*config.option.buildFlags:
SR*config.option.linkCommand:         cc -o
SR*config.option.instrumentC1:         True
SR*config.option.instruCommand:        ic9
SR*config.option.compilerFlags:       -c
SR*config.option.instrumentS1:         True
SR*config.option.instrumentS0:         False
```

This is the resource file for “C++” on the Sun platform.

```
! Flags needed to instrument C++ code under SunOS
! for compilation with CC

SR*config.option.runTarget:                a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:            True
SR*config.option.buildCommand:             make -f
SR*config.option.instrumentAnsiC:          False
SR*config.option.instrumentKRC:            False
SR*config.option.compilerCommand:         CC
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule:        crun4.o
SR*config.option.linkLibraryFlags:        -lm
SR*config.option.buildFlags:
SR*config.option.linkCommand:              CC -o
SR*config.option.instrumentC1:             True
SR*config.option.instruCommand:            icpp9
SR*config.option.compilerFlags:            -c -I/usr/
                                           lang/SC2.0.1/include/CC_413 -I/usr/lang/
                                           SC2.0.1/include/cc_413
SR*config.option.instrumentS1:             True
SR*config.option.instrumentS0:             False
```

**B.3.5 HP-UX**

This is the resource file for “C” on the HP-UX platform.

```
! Flags needed to instrument ANSI C code on the HP
! for compilation with cc
```

```
SR*config.option.instrumentS0:      False
SR*config.option.runTarget:         a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:     False
SR*config.option.buildCommand:      make -f
SR*config.option.linkCommand:       cc -o
SR*config.option.instrumentAnsiC:    False
SR*config.option.instrumentKRC:      True
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule: crun4.o
SR*config.option.linkLibraryFlags:
SR*config.option.compilerCommand:   cc
SR*config.option.buildFlags:
SR*config.option.instrumentC1:       True
SR*config.option.instruCommand:      ic9
SR*config.option.instrumentS1:       True
SR*config.option.compilerFlags:     -c
```



This is the resource file for “C++” on the HP-UX platform.

```
! Flags needed to instrument C++ code on the HP
! for compilation with CC
```

```
SR*config.option.instrumentS0:           False
SR*config.option.runTarget:              a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:          True
SR*config.option.buildCommand:           make -f
SR*config.option.linkCommand:            CC -o
SR*config.option.instrumentAnsiC:         False
SR*config.option.instrumentKRC:           False
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule:      crun4.o
SR*config.option.linkLibraryFlags:
SR*config.option.compilerCommand:        CC
SR*config.option.buildFlags:
SR*config.option.instrumentC1:            True
SR*config.option.instruCommand:           icpp9
SR*config.option.instrumentS1:            True
SR*config.option.compilerFlags:          -c
```

**B.3.6 RS-6000**

This is the resource file for “C” on the RS-6000 platform.

```
SR*config.option.buildCommand:make -f
SR*config.option.instrumentAnsiC:False
SR*config.option.instrumentKRC:True
SR*config.option.compilerCommand:cc
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule:crun4.o
SR*config.option.linkLibraryFlags:-lm
SR*config.option.buildFlags:
SR*config.option.instruCommand:ic9
SR*config.option.linkCommand:cc -o
SR*config.option.instrumentC1:True
SR*config.option.compilerFlags:-c -I/usr/include
SR*config.option.instrumentS1:True
SR*config.option.instrumentS0:False
SR*config.option.runTarget:a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:False
```

This is the resource file for “C++” on the RS-6000 platform.

```
SR*config.option.buildCommand:           make -f
SR*config.option.instrumentAnsiC:         False
SR*config.option.instrumentKRC:           False
SR*config.option.compilerCommand:         xlc
SR*config.option.runArgs:
SR*config.option.linkRuntimeModule:       crun4.o
SR*config.option.linkLibraryFlags:       -lm
SR*config.option.buildFlags:
SR*config.option.instruCommand:           icpp9
SR*config.option.linkCommand:             xlc -o
SR*config.option.instrumentC1:            True
SR*config.option.compilerFlags:          -c
SR*config.option.instrumentS1:            True
SR*config.option.instrumentS0:            False
SR*config.option.runTarget:              a.out
SR*config.option.linkFlags:
SR*config.option.instrumentC++:           True
```



# *cover*—TCAT C/C++’s Coverage Analyzer

This chapter explains options for invoking and customizing the “cover” coverage analyzer. This chapter applies to all editions of TCAT C/C++.

---

These are the options on how to invoke **cover**. This command, used inside the TCAT C/C++ graphical user interface, is used to produce a coverage report which, optionally, can report results in a Reference Listing. The Reference Listing report allows you to look up a segment in order to identify the actual unexecuted code, and plan new test cases.

## C.1 Command Line Invocation

The complete syntax for calls to **cover** is listed below. Items enclosed in [brackets] are to be included zero or more times.

```
cover [-switches] [tracefile]
[-a old-archive]
[-b file]
[-c]
[-C1]
[-d name [name[s]]]
[-DI deinsty-file]
[-DL]
[-f new-archive]
[-help]
[-h | -h name[s]]
[-H]
[-N]
[-n]
[-nl file]
[-NH]
[-NM]
[-m]
[-l | -l name[s]]
```

[-p]  
[-q]  
[-r file]  
[-S0]  
[-S1]  
[-s]  
[-SU]  
[-T [threshold]]  
[-w width]]  
[-Z reference listing]

The options may be used to vary the processing and reports generated by **cover**. The options are listed in alphabetical order.

**[tracefile [tracefile]]** These are the names of the trace files that you wish to process. If there are no trace files then **cover** looks for data in the default trace file name **Trace.trc**.

If there are no names given, and **Trace.trc** is not present then an error message is issued.

If there are multiple trace files, each trace file is processed in the order presented.

**Caution:** *The list of trace files must be the first set of arguments. The list is ended by the first symbol that appears with a '-', i.e. by the first optional switch.*

**-a old-archive**

*Old Archive File Name Switch.* You can include data from an old archive file in your reports. On the standard cumulative coverage report, this data will be included in the “Cumulative Summary” test results, but not under the column “Test”. To test iteratively, progressing through a structured series of tests towards higher C1 values, each run of **cover** should include the cumulative archive file from the previous test.

If you do not include an archive file, the “Cumulative Summary” figures will be the same as those for “Test”. Alternatively, if no -a option is given, the file Archive is used by default.

The -a option interacts with the other report options discussed below.

**-b file**

*Banner File Name Switch.* This allows you to include specific text, taken from the first line of the file named

	<i>title</i> as a title for your reports. A maximum of 80 characters is allowed for titles.
<b>-c</b>	<i>Cumulative Report Switch.</i> This option prints the Cumulative report only.
<b>-C1</b>	<i>Branch Coverage Reporting Switch.</i> Turns on reporting of C1 or branch coverage.  <b>Note:</b> Unless at least one of <b>-C1</b> , <b>-S1</b> , or <b>-S0</b> is turned on, no coverage report will be generated.
<b>-d name</b>	<i>Module Name Delete Switch.</i> If this switch is present then the named modules, if found in the current execution, are deleted from the generated Archive file. Subsequently, <b>cover</b> will never have heard about these names. This switch is useful in updating an extensive test record that would otherwise be lost due to the complexity of editing the Archive file.
<b>-DI deinst-file</b>	<i>De-instrument Switch.</i> Allows the user to specify a list of modules that are to be excluded from coverage reporting. Only the list of module names found in the specified <code>deinst-file</code> is to be excluded from coverage reporting. The module names can be specified in any format. White space (such as tabs, spaces) is ignored. <code>deinst-file</code> is also the file where new modules that pass the coverage threshold value (see the <b>-T</b> switch) will be written.
<b>-DL</b>	<i>De-instrument Module List Switch.</i> Allows the user to see which modules are excluded from coverage reporting. This switch is used along with the <b>-DI</b> switch. The list of excluded modules is printed at the end of the coverage report
<b>-f new-archive</b>	<i>New Archive File Name Switch.</i> Newly accumulated test coverage data will be placed in this file. If you do not include a different name with this switch, the accumulated test data will be placed in the default name Archive.

---

**Caution:** Each time you run **cover**, you will write over the contents of the Archive file unless you use the **-f** switch to direct the Archive file to another place. You may wish to remove the filename before starting a new test sequence.

---

<b>-help</b>	Print valid syntax
--------------	--------------------

<b>-h   -h [name]</b>	Linear Histogram Report Switch (-h).
<b>-l   -l [name]</b>	Logarithmic Histogram Report Switch (-l).
	These two options produce two “histogram” reports that graph the frequency distribution of the segments exercised in a single module. The histograms provide a module-by-module analysis of testing coverage, combining current trace file data with archive data included through the -a option or using the default Archive file. If the optional name argument is present, then the corresponding histogram for only the named module is produced; otherwise, <b>cover</b> produces histograms for all modules found. There can be multiple names in the argument if you want histograms of several modules. Also, the names can be mixed between linear and logarithmic histograms.
<b>-H</b>	<i>Hit Report Switch.</i> Lists the segments that have been hit one or more times in current or past tests. This report analyzes the cumulative effect of the current trace file and any archive data included through the use of the -a option or using the default Archive file.
<b>-m</b>	<i>Minimal Output Switch.</i> When present, <b>cover</b> suppresses banner information, list of current options and trace file descriptions. The coverage report contains only the reports requested.
<b>-N, -n</b>	<i>Not Hit Report Switch.</i> This option produces the “Not Hit” report which lists segments that have not been exercised. This report analyzes the cumulative effect of the current trace file and any archive data included through the use of the -a option or using the default Archive file.
<b>-NH</b>	<i>Newly Hit Report Switch.</i> Shows the segments by module that were hit in the current execution that were not hit previously. Thus this gives the user an assessment of the value of the most-recently added test(s). This shows what the current test “gained”. Output is the complement of the “Newly Missed” report.
<b>-nl file</b>	<i>Name List Switch.</i> This switch specifies that only the list of module names found in the specified <i>namefile</i> file is to be reported on in the current coverage report. Coverage on other module names that may appear in the archive or supplied trace files are ignored; however, the data is accumulated in the archive file.



The names used must be specified one name per line. White space (tabs, spaces, etc.) on the line is ignored.

The following reports are affected by the existence of a namefile:

- Cumulative Report
- Past Report
- Not Hit Report
- Hit Report
- Newly Hit Report
- Newly Missed Report.

The histogram outputs are not affected. There is a separate name mechanism that can be used to produce individual histogram reports.

<b>-NM</b>	<i>Newly Missed Report Switch.</i> This option produces the Newly Missed report. Shows which segments, by module, hit in any prior test that were not hit in the current test. This shows what the current test “lost”. This output is the complement of the Newly Hit report.
<b>-p</b>	<i>Past Report Switch.</i> Print only the Past Test report; this option should be used in conjunction with the -a option when you want to analyze the overall performance of a set of past tests.
<b>-q</b>	<i>Quiet Output Switch.</i> Suppress printout of current version and release information (this can be used to facilitate running <b>cover</b> in batch mode).
<b>-r report</b>	<i>Coverage Report File Name Switch.</i> Normally the report is written to the file Coverage (the default name), but you can rename the file with this switch. CAUTION: You will overwrite any file you name with this switch.
<b>-S1</b>	<i>Call-Pair Coverage Switch.</i> If present, the report will show call pair coverage.
<b>-S0</b>	<i>Module Coverage Switch.</i> If present, the report will show module coverage.

---

**NOTE:** Unless at least one of **-C1**, **-S1**, or **-S0** is turned on, no coverage report will be generated. However, not both **-S1** and **-S0** can be present; if they are then only **-S1** is assumed.

---

<b>-s</b>	<i>Sort Switch.</i> This option produces output reports with module names sorted alphabetically.
<b>-SU</b>	<i>Suppress Update Switch.</i> During processing, <b>cover</b> will suppress updating of the archive file, either the default Archive or the file named by the <b>-f</b> switch. <b>cover</b> will read the data in the archive file to form the basis for the “past test” information.
<b>-T threshold</b>	<i>Coverage Threshold Switch.</i> Threshold is a real number that specifies threshold value. Any module with a coverage percentage greater than or equal to this threshold value will be written to the de-instrumented file (see the <b>-DI</b> <i>deinst-file</i> switch). If no threshold is specified, then the default value of 85 percent is assumed.
<b>-w width</b>	<i>Report Width Switch.</i> Normally the reports generated by <b>cover</b> are wide enough to accommodate module names up to 21 characters in length. The internal limit on name length is, however, 128 characters. You can use this switch to force <b>cover</b> system to generate reports that are wide enough to accommodate the full 128 character module names.  The width factor is the number of additional characters to be added to the report. The default value is zero. Maximum width is $128 - 21 = 107$ . <b>WARNING:</b> Reports with high values for the <b>-w</b> option may contain long lines and may not be suitable for printing directly.
<b>-Z reference</b>	<i>Annotated Reference Listing Switch.</i> <b>cover</b> will analyze the specified archive file, and any specified trace files, and will produce a report that shows the coverage level achieved for all modules that are named in the specified reference listing. The reference listing must be one that is produced by a current release of the <i>TCAT C/C++</i> instrumentor. Reference listings produced by earlier versions may not necessarily work correctly with this switch.  If a module is tested but the name is not found in the supplied reference listing, then that coverage is not reported. Similarly, if a name appears in the reference listing and is not one that exists in the archive file, no coverage will be reported.

### **C.1.1      Error Processing**

In case there is an error, **cover** gives a response line (usage line) indicating the set of switches and options. This response is the same as the **-help** response.



# Index

- 
- A**
    - annotating calltrees 83–87
    - archive files 53, 54, 58, 89
      - overwriting 53
  - B**
    - branch coverage, C1 2, 3, 5, 37, 57, 84
  - C**
    - C1 expansion tree 57
    - call pair coverage, S1 2, 3, 5, 7, 57, 84
    - caller-to-callee connections 78
    - calltree 4, 7
    - calltree display options 72–74
    - calltree graph
      - basename.cg 18
    - cover, coverage analyzer 147–153
      - command line syntax 147
      - invoking 147
      - reports
        - generating 26
        - viewing 26–27
    - cyclomatic complexity 4, 84, 107
  - D**
    - database reference file 18
    - dbx debugger 46
    - directed graph 5, 18
  - E**
    - equivalence class coverage, Ct 2, 3, 4, 5, 7
  - F**
    - font
      - italics xvii
      - italix xvii
    - font, bold face xvii
    - font, courier xvii
  - I**
    - instrumenting an application 18, 19
    - instrumentor engine 117–129
      - C (ic) and C++ (icpp) versions 10, 37, 49, 117
      - instrumentor switches 119–121
  - L**
    - Link/Build/Run Options 20
    - load setting 17, 35, 131
  - M**
    - make files 45
    - manual organization xvi
    - motifbur.uil 19
    - Motifburger 13–30
  - O**
    - object files 18, 45
  - P**
    - PostScript (.ps) file 82
  - R**
    - regression testing 1
    - resource files 131–145
      - definitions of variables 131–133
      - platforms
        - DEC Alpha 134
        - HP-UX 142
        - RS-6000 144
        - SGI 136
        - Solaris 138
        - Sun 140
    - rm command 46
    - runtime object module 20, 45
      - levels of buffering 51
      - linking 21, 41
      - prompts 22
      - selecting 20, 39
      - setup options 49–51
-

---

## INDEX

---

### S

S1 expansion tree 57  
SCCS delta command 47  
SCCS edit command 47  
SCCS get command 47  
static analysis 1  
S-TCAT 4, 31

### T

#### TCAT

- building 45
- database directory 118, 123–129
- file filter 42–43
- instrument/compile option 46
- instrument/compile options 36–37
- instrumenting 45
- instrumenting an application 45
- invoking 14–16, 31
- kill button 48
- link/build/run options 38–41
- linking 45
- main window 32–48
- saving settings 35
- selecting database name 35
- selecting files 33
- selecting utilities 33
- shell 33, 47
- tutorial 13–30

tcat\_db directory 18  
TCAT-PATH 2, 4, 7  
test coverage 1  
text

- "double quotation marks" xvii
- boldface xvii
- italics xvii

text, boldface xvii  
text, courier xvii  
text, italix xvii  
trace files 25, 53, 58  
T-SCOPE 4, 5  
tutorial 13–30

### V

vi editing program 46

### X

X Window System 14  
Xcalltree

- invoking 60

Xcalltree Annotations 87  
Xcalltree utility 4, 10, 46, 59–87

- annotation 63, 65
- display options 62, 68, 72–74
- displaying a calltree 65

- displaying statistics 62, 77–78
- file format 59
- file selection 62, 64–67
- help 63
- main window 61–65
  - setting archive file 65
- print options 62, 79–82
- root selection 70–71
- viewing source code 62, 75
- zoom 62, 69, 74

Xcover utility 8, 10, 24, 28, 29, 45, 53–58

- invoking 54
- project files 58
- viewing source code 28

Xdigraph utility 10, 46, 89–116

- annotation 112–116
- display options 99–103
- displaying source code 105
- displaying statistics 106–107
- file selection 95–98
- invoking 90
- main window 92–94
- print options 108–111
- zoom 100, 104

Xtcat 15