

USER'S GUIDE

TCAT for Java™

Version 1.0

Test Coverage
Analysis Tool
For Java™



SOFTWARE RESEARCH, INC.

This document property of:

Name: _____

Company: _____

Address: _____

Phone _____



SOFTWARE RESEARCH, INC.

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 © 1997 by Software Research, Inc

(Last Update February 3, 1998)

[/coverage/97covbook/coverage.book/tcatJava/tcat9.Java.book](#)

Table of Contents

	List of Figuresxi
	Prefacexiii
CHAPTER 1	Overview of TCAT for Java	1
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	6
1.3	TCAT for Java	7
1.3.1	Feature Summary	8
	Instrumentor	8
	Runtime	8
	Coverage	8
1.3.2	File Inventory	9
1.3.3	Installation Process	10
CHAPTER 2	Quick Start	11
2.1	Overview	11
2.2	STEP 1: Starting Up TCAT for Java	12
2.3	STEP 2: Selecting Load Setting	14
2.4	STEP 3: Instrumenting the Example Applet	15
2.5	STEP 5: Running the Applet	18

TABLE OF CONTENTS

2.6	STEP 6: Opening the Coverage Window	.19
2.7	STEP 6: Choosing a Trace File	.20
2.8	STEP 8: Generating and Viewing a Report	.21
2.9	STEP 9: Viewing a Logical Branch's Source Code	.23
2.10	STEP 10: Sign Off and Cleanup	.24
2.11	Summary	.24

CHAPTER 3 TCAT for Java™
Graphical User Interface25

3.1	TCAT for Java	.25
3.2	Invoking TCAT for Java	.25
3.3	TCAT for Java Main Window	.26
3.3.1	File	27
3.3.2	Options	27
3.3.3	File Selection Dialog Window	27
3.3.4	Utilities Section Dialog Window	27
3.3.5	Shell Section	27
3.4	File Pull-Down Menu	.28
3.4.1	Load Setting	29
3.4.2	Save Setting	29
3.4.3	Set Project DB	29
3.4.4	Exit	29
3.5	Options Pull-Down Menu	.30
3.5.1	Instrumentor/Compiler Options	30
	Instrumentor Options	31
	Compiler Options	31
3.6	Link/Build/Run Options	.32
3.6.1	Build Options	33
	Command	33
	Flags	33
3.6.2	Run Options	34
	Target Name	34
	Command Line Args	34
3.7	File Selection Section	.35
3.7.1	Directories	36
3.7.2	Files	36
3.7.3	Selection	36
3.7.4	Clear All	36
3.7.5	Select All	36
3.8	Utilities Section	.37

3.8.1	Instrument	38
3.8.2	Build	38
3.8.3	Run	38
3.8.4	Build & Run	38
3.8.5	Coverage	38
3.8.6	Xdigraph	38
3.8.7	Xcalltree	38
3.8.8	Vi	38
3.8.9	Rm	38
3.8.10	Appletviewer	39
3.8.11	Appletviewer -debug	39
3.9	Shell Section	40
3.9.1	Command	40
3.9.2	Message Area	40
3.9.3	Kill	40
CHAPTER 4	Runtime Classes	41
4.1	TCAT for Java “Runtime” Support Options	41
4.2	Performance Gain With Buffering	43
CHAPTER 5	Xcover — Coverage Analyzer	45
5.1	Xcover	45
5.2	Xcover Functionality	45
5.3	Command Line Invocation	46
5.4	Xcover Sample Screens	47
5.5	Operation of Xcover	48
5.5.1	Xcover Options	49
5.5.2	Selecting a Trace File	49
5.5.3	Selecting an Archive File	49
CHAPTER 6	Xcalltree	51
6.1	Purpose	51
6.2	Xcalltree File Format	51
6.3	Invoking Xcalltree	52
6.4	Xcalltree Main Window	53
6.4.1	File	54
6.4.2	Options	54
6.4.3	Zoom In & Zoom Out	54
6.4.4	View Source	54

TABLE OF CONTENTS

6.4.5	Statistics	54
6.4.6	Print	54
6.4.7	Annotation.....	55
6.4.8	Help	55
6.5	File Pull-Down Menu	56
6.5.1	Load New Graph.....	57
6.5.2	Load New Multi Graph	57
6.5.3	Set Archive	57
6.6	Calltree File Selection Dialog Box	58
6.6.1	Filter	59
6.6.2	Directories.....	59
6.6.3	Files	59
6.6.4	Selection.....	59
6.6.5	OK.....	59
6.6.6	Filter Button	59
6.6.7	Cancel.....	59
6.7	Option Window	60
6.7.1	Zoom Scale.....	61
6.7.2	Horizontal Spacing	61
6.7.3	Depth	61
6.7.4	Root Name	62
6.7.5	Edge Characteristics	64
	Edge Color	64
	Unhighlighted Edge	64
	Display Mode	64
6.7.6	Node Characteristics	65
	Size	65
	Aspect Ratio	65
	Default Color	65
	Low-level Color	65
	Normal Color	65
	High-level Color	65
	Apply	65
6.8	Zoom In & Zoom Out Options	66
6.9	View Source Window	67
6.9.1	Description of Source Code Viewing	67
6.10	Statistics Window	68
6.10.1	Links	69
6.10.2	Call pairs.....	69
6.10.3	Modules/Depth	69
6.10.4	Recursive	69
6.11	Print Window	70
6.11.1	Paper Size Information	71
6.11.2	Enlargement Factors.....	72

6.11.3	Font Information	.73
6.11.4	Print Locator	.73
6.12	Annotation Window	.74
6.12.1	Threshold 1 & Threshold 2	.75
6.12.2	None	.75
6.12.3	S0	.75
6.12.4	Ninvokes	.75
6.12.5	S1	.75
6.12.6	C1	.75
6.12.7	Cyclo	.75
6.12.8	Nsegs	.75
6.12.9	Npairs	.76
6.12.10	Nlines	.76
6.12.11	Npaths	.76
6.12.12	User	.76
6.12.13	Connections	.76
6.12.14	Apply	.76
6.12.15	Reset	.76
6.12.16	Close	.77
6.12.17	Help	.77
6.13	Quick Reference Guide to Xcalltree Annotations	.78
CHAPTER 7	Xdigraph	.79
7.1	Purpose	.79
7.2	Xdigraph File Format	.79
7.3	Invoking Xdigraph	.80
7.4	Xdigraph Main Window	.82
7.4.1	File	.83
7.4.2	Options	.83
7.4.3	Zoom In	.83
7.4.4	Zoom Out	.83
7.4.5	View Source	.83
7.4.6	Statistics	.83
7.4.7	Print	.83
7.4.8	Annotation	.84
7.4.9	Help	.84
7.5	File Pull-Down Menu	.85
7.5.1	Load New Graph	.86
7.5.2	Load New Module	.86
7.5.3	Set Archive	.86
7.5.4	Exit	.86
7.5.5	Digraph File Message Box	.87
	Filter	.88
	Directories	.88

TABLE OF CONTENTS

	Files	88
	Selection	88
	OK	88
	Filter Button	88
	Cancel	88
7.6	Options Window	89
7.6.1	Zoom Scale	90
7.6.2	Node Characteristics	91
	Shape	91
	Size	91
	Vertical Spacing	91
	Aspect ratio	91
7.6.3	Edge Characteristics	92
	Unhighlighted Edge	92
	Eccentricity	92
	Default Color	92
	Low-level Color	92
	Normal Color	92
	High-level Color	92
	Apply	93
	Reset	93
	Close	93
	Help	93
7.7	Zoom In/Zoom Out Window	94
7.8	View Source Window	95
7.9	Statistics Window	96
7.9.1	File Name	97
7.9.2	Node and Edge Count	97
7.9.3	Cyclomatic Number (Cyclomatic Complexity)	97
7.9.4	Average, Minimum and Maximum Path Lengths	97
7.9.5	Path Count by Iteration Groups	97
7.10	Print Window	98
7.10.1	Paper Size Information	99
7.10.2	Enlargement Factors	100
7.10.3	Font Information	101
7.10.4	Print Locator	101
7.11	Annotation Window	102
7.11.1	Threshold 1 & 2	103
7.11.2	None	103
7.11.3	Nhits	103
7.11.4	N%	103
7.11.5	Nlines	103
7.11.6	User	103

7.11.7	Highlight	104
7.11.8	Path File	104
7.11.9	Apply	104
7.11.10	Reset	104
7.11.11	Close	104
7.11.12	Help	104
7.11.13	Colors	105
7.12	Quick Reference Guide to Xdigraph Annotations	106
APPENDIX A	Resource File Variables	107
APPENDIX B	cover9 —TCAT for Java's Coverage Analyzer	111

TABLE OF CONTENTS

List of Figures

FIGURE 1	TCAT Graphical Displays	5
FIGURE 2	Setting Up the Display (the invocation window).....	12
FIGURE 3	The TCAT for Java Main Window	13
FIGURE 4	Selecting the Load Setting	14
FIGURE 5	Instrumenting the Source File	16
FIGURE 6	Selecting the Runtime Object Module.....	17
FIGURE 7	Running TicTacToe	18
FIGURE 8	The Coverage Window	19
FIGURE 9	Selecting a Trace File Name	20
FIGURE 10	Coverage Information for Each Function	21
FIGURE 11	Looking at Source Code	23
FIGURE 12	TCAT for Java main window	26
FIGURE 13	File pull-down menu	28
FIGURE 14	Instrumentor/Compiler Options window	30
FIGURE 15	Link/Build/Run Options window	32
FIGURE 16	File Selection section of TCAT main window	35
FIGURE 17	Utilities section of TCAT for Java main window	37
FIGURE 18	Shell section of the TCAT window	40
FIGURE 19	Xcover Example Output 1	47
FIGURE 20	Xcalltree main window	53
FIGURE 21	File Pull-Down Menu	56
FIGURE 22	Calltree File Selection Dialog Box	58
FIGURE 23	Option window.....	60
FIGURE 24	Root Name Selection window example 1	62
FIGURE 25	Root Name Selection Window Example 2.....	63
FIGURE 26	Zoom In Option illustrated	66

LIST OF FIGURES

FIGURE 27	View Source Window	67
FIGURE 28	Statistics Window	68
FIGURE 29	Print Window	70
FIGURE 30	Annotation Window	74
FIGURE 31	Program edges as represented in a digraph.	81
FIGURE 32	Xdigraph main window	82
FIGURE 33	Digraph File Pull-Down Menu	85
FIGURE 34	Digraph File Message Box	87
FIGURE 35	Xdigraph Options Window	89
FIGURE 36	Zoom In feature illustrated	94
FIGURE 37	View Source Option Window	95
FIGURE 38	Statistics Option Window	96
FIGURE 39	Print Dialog Window	98
FIGURE 40	Annotation Thresholds Window.	102
FIGURE 41	Sample Annotation for User Threshold.	105

Preface

How this user's guide is organized.

Congratulations!

By choosing the TestWorks integrated 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 becoming more 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 for Java™ 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 call trees visualize the location, allowing you to make immediate improvements to the structure and performance of your software.

TestWorks for Java is the most complete solution available, with full-featured regression testing, coverage analyzers, and metric tools.

Audience

This manual is intended for software testers who are using **TCAT for Java**.

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

Typefaces

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

PREFACE

Overview of TCAT for Java

This chapter is an introduction to test coverage tools and TCAT for Java™, including TCAT for Java's new features and enhancements. TCAT for Java combines the functionality of previous versions of TCAT and S-TCAT. This chapter applies to all editions of TCAT for Java.

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 %.

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 method invocation pair (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 % of all modules).

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 "JAVA":

```
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**), method invocation pair (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 method invocation 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 the Java Compiler. 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 Java, 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 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 method invocation pair, or S1, metric with source instrumentation. TCAT is available for most Java compilers.

TCAT 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 method invocation pairs hit, method invocation pairs not-hit, method invocation pairs newly hit, method invocation 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 method invocation pair (call pair) structure — the “call tree” of the system being analyzed. A utility called **Xcalltree** allows the tester to analyze the call tree.

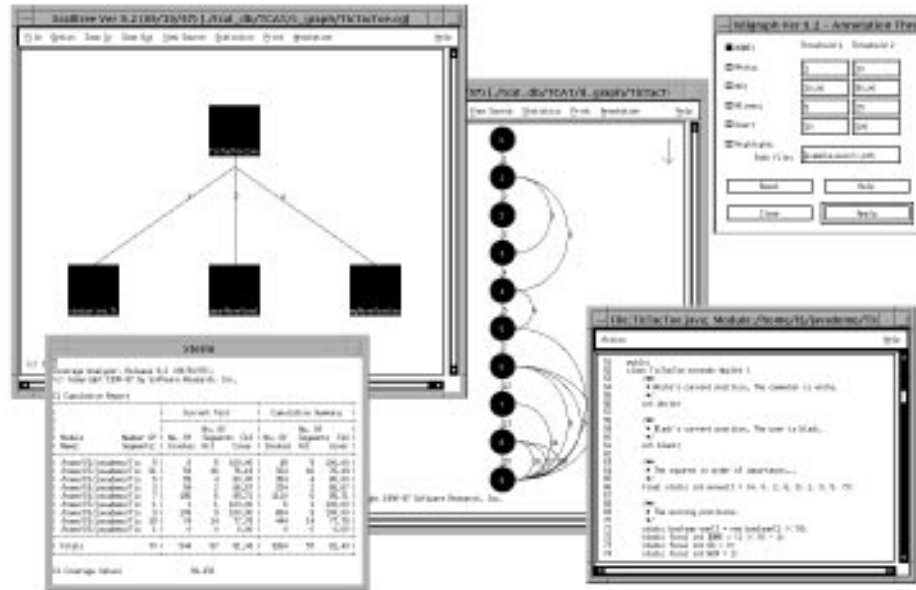


FIGURE 1 TCAT Graphical Displays
STW/Coverage dynamically generates a program's call-tree (top left), a module's directed graph (center), and a static report of the unexercised logical branches (bottom left) with source code displayed for an un-hit logical branch.

1.2.4 Effectiveness Assessment

Studies show that branch coverage of 85% or better tends to identify twice the number of defects that would have been found by “intuitive” testing. Method invocation pair (call pair) coverage of 90 % or better will detect 25% 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 method invocation pair (call pair) coverage on the entire system, e.g. including those with over 1 million LOC to process at one time. The method invocation pair coverage tends to identify interface errors — which are almost always self-evident when the caller-callee relationship is actually exercised. In fact, method invocation pair coverage reduces interface defects by a factor of three to five. Method invocation pair coverage is about 20% less difficult to obtain per KLOC than branch coverage.

For critical modules — usually a selected ten to fifteen % 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 % or better have been observed, even with the **Ct** coverage limited to about 75 %. It should be noted that a 100 % 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 for Java

TCAT for Java 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 enhancements to **TCAT for Java**, have been added to previous versions.

1.3.1.1 Instrumentor

- Java consolidated processing
- C1 and S1 consolidated coverage
- Enhanced error recovery during instrumentation
- High-speed, high-capacity system
- Instrumentor database available as a Java structure
- “Invisible” instrumentor operation as a wrapper on Java
- 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

- New **Xcover** interactive coverage analyzer

1.3.2 File Inventory

The following is an inventory of the files supplied with the program:

<i>ijava</i>	This file represent the new combined Java Instrumentor. It processes Java source code. Ijava will call the Java compiler and pass through all compiler switches that are supplied by the user. Control of the instrumentation process with command line switches requires prefixing <code>-TCAT</code> to all switches specific to the instrumentation process.
<i>jrunN.class</i>	The <i>ijava</i> runtime support classes, where <i>N</i> equals the level of buffering supported. The attached documentation gives the full description of the capabilities of the runtime system.
<i>cover</i>	This version of cover reads the new tracefile format.
<i>Xcover</i>	This is the new interactive GUI-based coverage analyzer.
<i>Xtcat</i>	This is the new GUI that provides access to the various utilities.
<i>Xcalltree</i>	This program draws calltrees based on information from the tracefiles and archive files produced by instrumentation.
<i>Xdigraph</i>	This program draws directed graphs based on information from the tracefiles ad archive files produced by instrumentation.

1.3.3 Installation Process

Installation instructions are provided in the “Open Me First” packet shipped with **TCAT for Java**.

Quick Start

This chapter explains getting started with **TCAT for Java™** using a demonstration test case. This chapter applies to all editions of **TCAT for Java**.

2.1 Overview

This application note will familiarize you with the main activities involved in using **TCAT for Java**, including instrumenting, compiling and running the target application or applet, and finally, looking at resulting coverage reports, calltree graphs and digraphs.

The program used to illustrate the operation of **TCAT for Java** is *Tic Tac Toe*, which allows you to play a game of Tic Tac Toe against the computer. By playing the game of Tic Tac Toe in this instrumented applet, you exercise various logical branches or segments, creating trace files from which the coverage reports are generated.

If you are a first-time **TCAT for Java** user, this chapter is best used if you refer to the other chapters in this manual 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 for Java

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 for Java** invocation window.

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

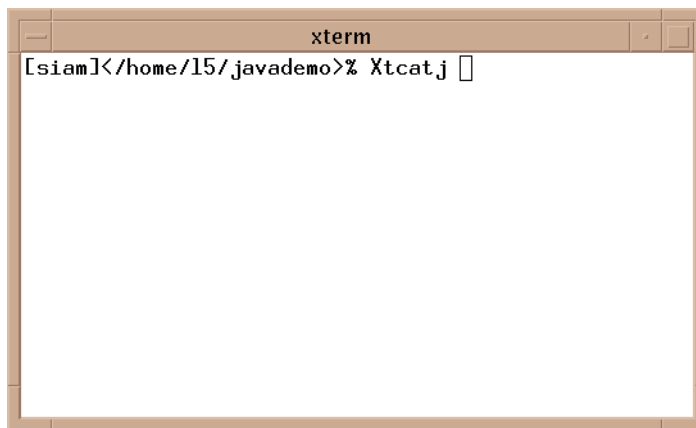


FIGURE 2 Setting Up the Display (the invocation window)

To invoke **TCAT for Java** and set up the display:

1. Position the mouse pointer in the invocation window. Activate the window by clicking the mouse pointer on it.
2. To invoke **TCAT**, type in
`Xtcat.j <CR>`
3. When you type in this command, the **TCAT for Java** main window pops up.
4. To terminate **TCAT for Java**, click on the **Files** pull-down menu from the main window and select **Exit**.

After **TCAT for Java** is loaded, you will see this panel:

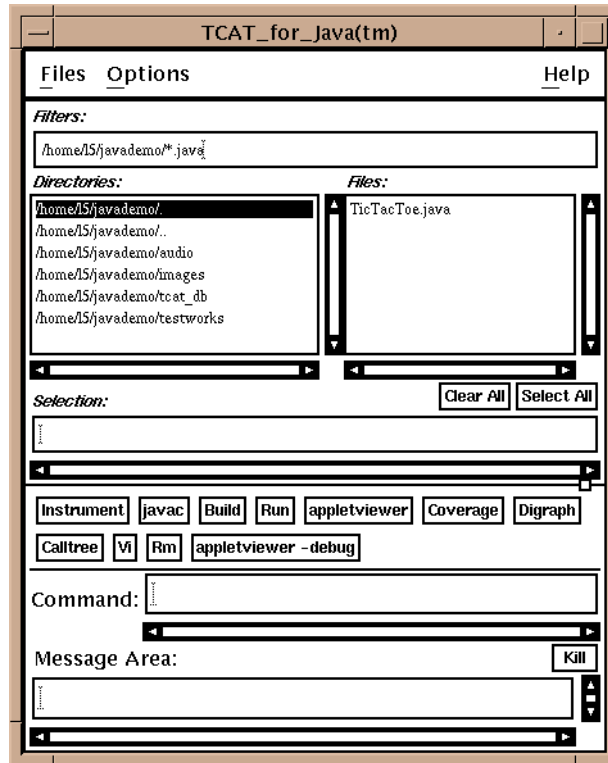


FIGURE 3 The TCAT for Java Main Window

2.3 STEP 2: Selecting Load Setting

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

Select the resource file you wish to use.

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

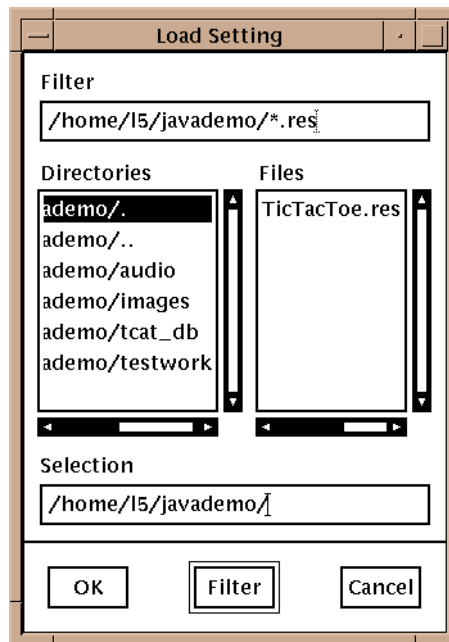


FIGURE 4 Selecting the Load Setting

2.4 STEP 3: Instrumenting the Example Applet

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

Instrument *TicTacToe* 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 applet 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.
- *mdf.pro* — A profile of the applet for use on your Web server.

TCAT for Java puts the first three of the files listed above into the *tcat_db* directory. It places the *mdf.pro* file in the current working directory.

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

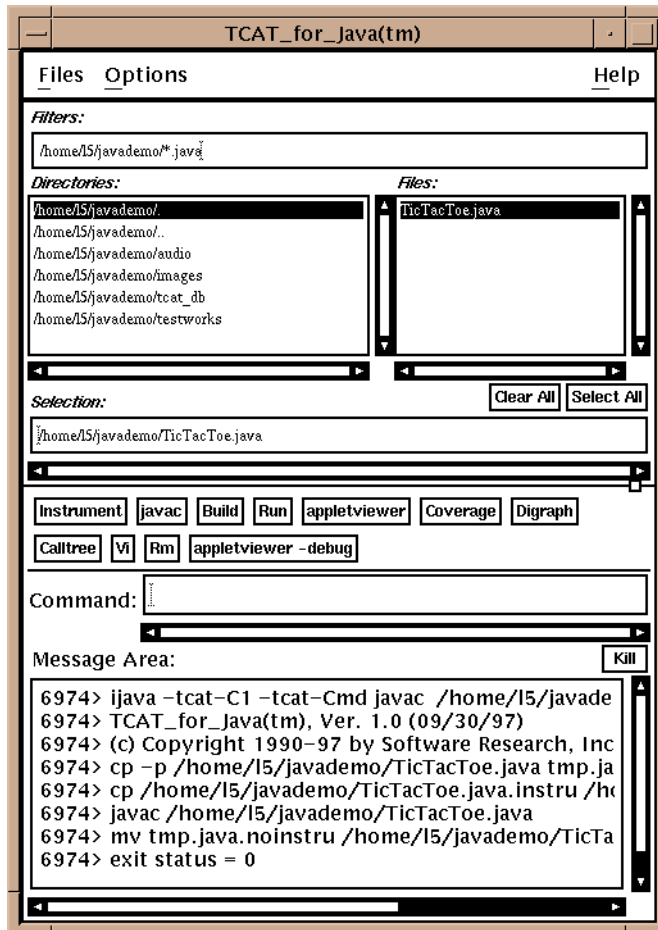


FIGURE 5 Instrumenting the Source File

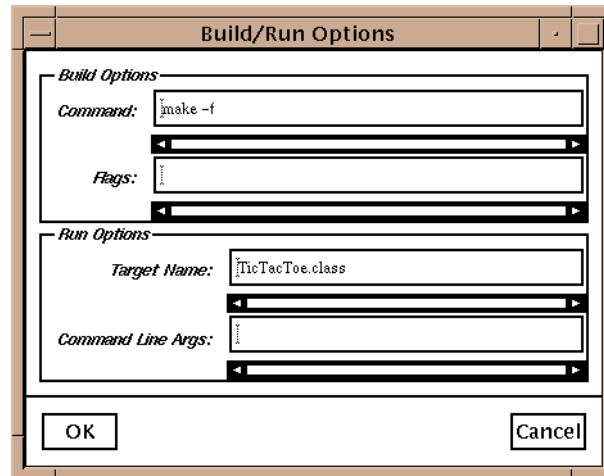


FIGURE 6 Selecting the Runtime Object Module
[[REWRITE]]

The next step is to link the selected runtime object module with the instrumented applet's object modules, named *example.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 *example.o*.
3. To link, click on the **Link** button.

2.5 STEP 5: Running the Applet

During instrumentation, **TCAT for Java** inserted function calls at each logical branch it found. In order to later determine the C1 coverage, you must run the applet.

By running *TicTacToe* and playing the game, you are exercising segments of the *TicTacToe* 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 applet:

1. Use the filter to select the **.html* files.
2. Click the **Appletviewer** or **Appletviewer -debug** button.
3. The appletviewer and *TicTacToe* applet will appear. Play the game.
4. When you are finished playing, click the **Applet** button and choose "Quit".

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

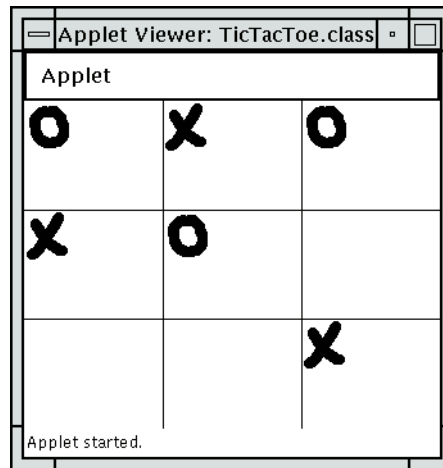


FIGURE 7 Running *TicTacToe*

2.6 STEP 6: Opening the Coverage Window

All the information from the run of the applet 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* invocation window's **Coverage** button.
2. The **Xcover** window pops up.
3. Use the mouse to drag the window below the **TCAT for Java** invocation window.

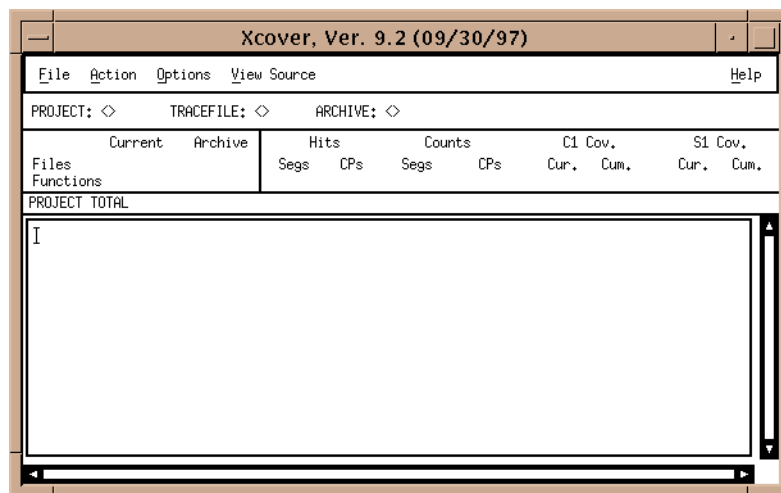


FIGURE 8 The Coverage Window

2.7 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:

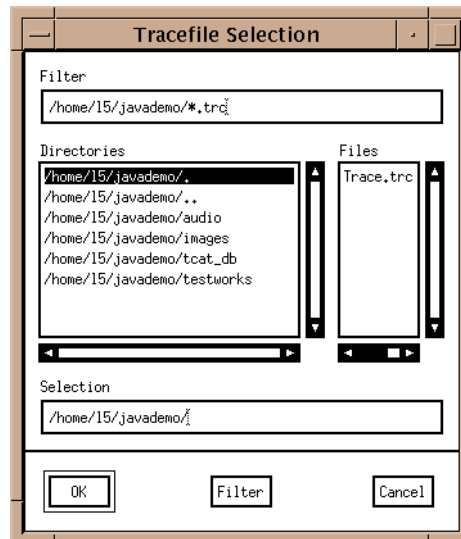


FIGURE 9 Selecting a Trace File Name

2.8 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 <TicTacToe> line to expand the display to include information on each function.

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

Hits	The number of times the element was executed during the test.
Count	The total number of segments within the function.
C1	The percentage of branch coverage for each function.
S1	The percentage of method invocation pair (call pair) coverage for the function.

	Current	Archive	Hits		Counts		C1 Cov.		S1 Cov.	
	Files	Modules	Segs	CPs	Segs	CPs	Cur.	Cum.	Cur.	Cum.
PROJECT TOTAL	1	0	54	0	70	6	77.14	77.14	0.00	0.00
</home/15/javademo/TicTacToe>			54	0	70	6	77.14	77.14	0.00	0.00
<TicTacToe::isNon(void>>			5	0	5	0	100.00	100.00	100.00	100.00
Segment 1			8							
Segment 2			4088							
Segment 3			8							
Segment 4			504							
Segment 5			3584							
<TicTacToe::bestMove(int,int)>			16	0	21	0	76.19	76.19	100.00	100.00
<TicTacToe::yourMove(boolean,i)>			4	0	5	0	80.00	80.00	100.00	100.00
<TicTacToe::myMove(boolean)>			2	0	3	1	66.67	66.67	0.00	0.00
<TicTacToe::status(int)>			5	0	7	0	71.43	71.43	100.00	100.00
<TicTacToe::init(void)>			1	0	1	0	100.00	100.00	100.00	100.00
<TicTacToe::paint(void)>			9	0	9	0	100.00	100.00	100.00	100.00
<TicTacToe::mouseUp(boolean)>			12	0	18	5	66.67	66.67	0.00	0.00
<TicTacToe::getAppletInfo(int)>			0	0	1	0	0.00	0.00	0.00	0.00

FIGURE 10 Coverage Information for Each Function

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

4. Click on the `<TicTacToe::isWon(viod)>` line to expand the display.

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

2.9 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 left mouse button.
9. **TCAT for Java** 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:

```

72     static final int DONE = (1 << 9) - 1;
73     static final int OK = 0;
74     static final int WIN = 1;
75     static final int LOSE = 2;
76     static final int STALEMATE = 3;
77
78     /**
79      * Mark all positions with these bits set as winning.
80      */
81     static void isWon(int pos) {
82         for (int i = 0 ; i < DONE ; i++) {
S:4  ->85         if ((i & pos) == pos) {
84             won[i] = true;
85         }
S:5  ->86     }
S:3  ->87     }
88
89     /**
90      * Initialize all winning positions.
91      */
92     static {
93         isWon((1 << 0) | (1 << 1) | (1 << 2));
94         isWon((1 << 3) | (1 << 4) | (1 << 5));
95         isWon((1 << 6) | (1 << 7) | (1 << 8));

```

FIGURE 11 Looking at Source Code

2.10 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 for Java** invocation window by clicking on the **File** pull-down menu and selecting **Exit**.

2.11 Summary

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

TCAT for Java™

Graphical User Interface

This section describes the TCAT for Java graphical user interface (GUI) for both the Standard and Professional editions of the product.

3.1 TCAT for Java

The new **TCAT for Java** graphical user interface (GUI) allows you to instrument, compile, run, test, debug, build and edit your Java application or Applet, as well as providing point-and-click access to reports, directed graphs and calltrees.

3.2 Invoking TCAT for Java

TCAT for Java can be invoked in one of two ways.

(1) from the command line with the command

`xtcattj`

(2) from the STW suite's main screen, by clicking the **TCAT for Java** icon.

3.3 TCAT for Java Main Window

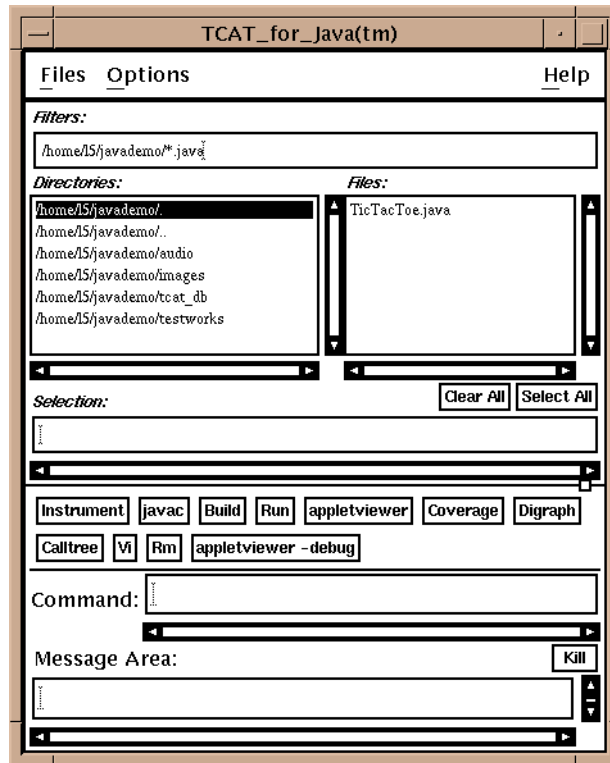


FIGURE 12 TCAT for Java main window

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, build and run your application.

3.3.3 File Selection Dialog Window

This window presents the available files when you choose to open a file from the File menu.

3.3.4 Utilities Section Dialog Window

This window 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

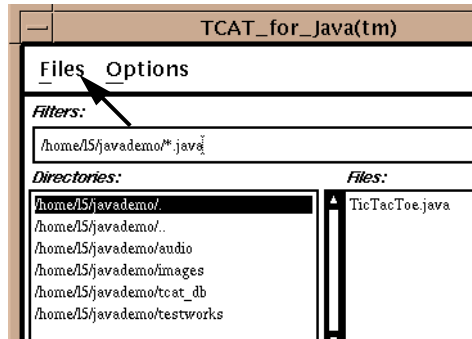


FIGURE 13 File pull-down menu

3.4.1 Load Setting

Lets you load previously saved settings for the various **TCAT for Java** options.

3.4.2 Save Setting

Lets you save settings for the various **TCAT for Java** 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 for Java**, 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.

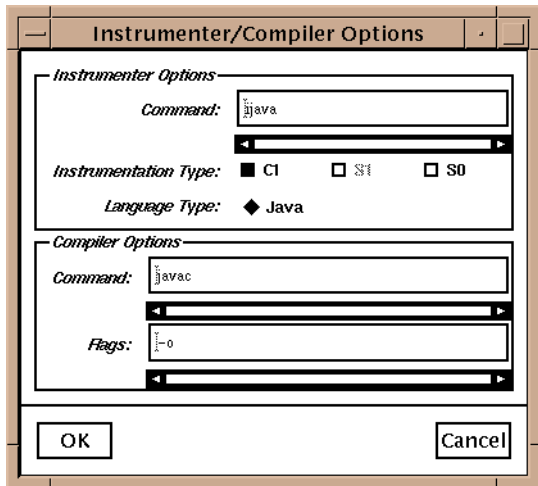


FIGURE 14 Instrumentor/Compiler Options window

3.5.1.1 Instrumentor Options

Command

Enter `ijava` to select the appropriate instrumentor.

Instrumentation Type

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

3.5.1.2 Compiler Options

Compiler

Enter the name of the desired Java compiler.

Flags

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

3.6 Link/Build/Run Options

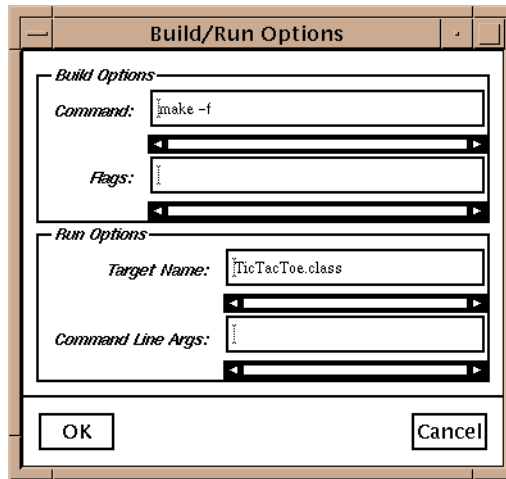


FIGURE 15 Link/Build/Run Options window

3.6.1 Build Options

3.6.1.1 Command

This option specifies the command used when you select Build.

3.6.1.2 Flags

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

3.6.2 Run Options

3.6.2.1 Target Name

The target name is dependent on whether you are building an application or an applet.

Enter your application class name if you will be running an application.

Enter the name of the HTML file that involves your applet if you are running an applet.

3.6.2.2 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 *.java in the Filters box, all files with the extension.out would appear in the Files box.

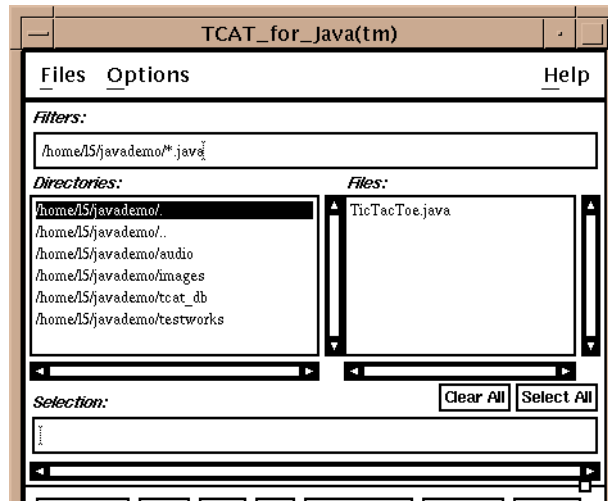


FIGURE 16 File Selection section of TCAT main window

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 filtered subset thereof) 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. You 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

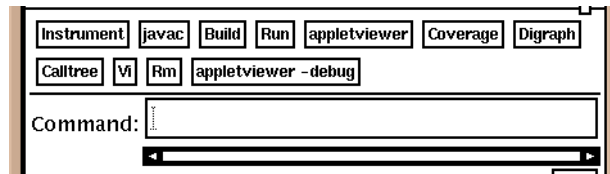


FIGURE 17 Utilities section of TCAT for Java main window

The Utilities section contains the following buttons:

3.8.1 Instrument

Select a Java source code and click this button to instrument it.

3.8.2 Build

This option allows you to build the target application. It uses command specified in the Build/Run options dialog.

3.8.3 Run

Select a Java application (Not an applet), and click the Run button to execute it using the Java command.

3.8.4 Build & Run

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

3.8.5 Coverage

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

3.8.6 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.

3.8.7 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.

3.8.8 Vi

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

3.8.9 Rm

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

3.8.10 Appletviewer

This button involves the appletviewer on the selected HTML file.

3.8.11 Appletviewer -debug

This button involves the appletviewer in debug mode on the selected HTML file.

3.9 Shell Section

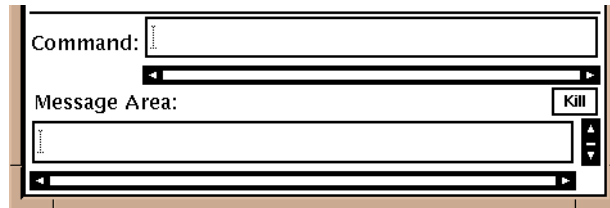


FIGURE 18 Shell section of the TCAT window

3.9.1 Command

Entries in this field will be passed on to the shell.

3.9.2 Message Area

Messages from the shell regarding **TCAT for Java** 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 Classes

This chapter describes TCAT for Java's™ Runtime Options and applies to all editions of the product.

4.1 TCAT for Java “Runtime” Support Options

Provided with TCAT for Java is a package containing the TCAT for Java runtime classes.

By default, this package is installed into the `$$SR/lib` directory. The actual classes in the package are installed into `$$SR/lib/testworks/runtime`.

Your instrumented Java classes must be able to find this package. Therefore, during instrumentation and execution your `CLASSPATH` environmental variable must point to the top of this package.

Inside the `$$SR/lib/testworks/runtime` directory are found several classes that implement various levels of trace buffering.

The default level of buffering is one. This is effectively no buffering, since as each trace hit occurs it is written to the trace file. To increase the size of buffering, copy the desired class from the chart below to `jrun.class` before executing your instrumented applet or application.

Class Name	Buffering level
jrun1.class	None
jrun100.class	100 Trace Hits
jrun10000.class	10000 Trace Hits
jrunInt.class	Infinite, Trace hits are not written to trace file until termination of application.

TABLE 1 Buffering level for runtime class

Note that if your applet or application terminates abnormally that up to one buffer full of trace data will be lost.

4.2 Performance Gain With Buffering

The larger the trace buffer the better the performance of your instrumental application.

Xcover — Coverage Analyzer

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

5.1 **Xcover**

Xcover analyzes the trace files created when an instrumented applet or application 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 for Java** 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.

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.

```
Xcoverj 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 TicTacToe example in various stages of expansion and contraction of the display.

Xcover, Ver. 9.2 (09/30/97)										
File Action Options View Source Help										
PROJECT: <TCAT>		TRACEFILE: <Trace.trc>		ARCHIVE: <>						
Files	Current	Archive	Hits		Counts		C1 Cov.		S1 Cov.	
Modules	9	0	Segs	CPs	Segs	CPs	Cur.	Cum.	Cur.	Cum.
PROJECT TOTAL			54	0	70	6	77.14	77.14	0.00	0.00
</home/15/javademo/TicTacToe>			54	0	70	6	77.14	77.14	0.00	0.00
<TicTacToe::isWon(void)>			5	0	5	0	100.00	100.00	100.00	100.00
Segment 1			8							
Segment 2			4088							
Segment 3			8							
Segment 4			504							
Segment 5			3584							
<TicTacToe::bestMove(int,int)>			16	0	21	0	76.19	76.19	100.00	100.00
<TicTacToe::yourMove(boolean,i)>			4	0	5	0	80.00	80.00	100.00	100.00
Segment 1			75							
Segment 2			0							
Segment 3			75							
Segment 4			19							
Segment 5			56							
<TicTacToe::myMove(boolean)>			2	0	3	1	66.67	66.67	0.00	0.00
Segment 1			56							
Segment 2			0							
Segment 3			56							
Callpair 1				0						
<TicTacToe::status(int)>			5	0	7	0	71.43	71.43	100.00	100.00
Segment 1			207							
Segment 2			42							
Segment 3			165							
Segment 4			0							
Segment 5			165							
Segment 6			0							
Segment 7			165							
<TicTacToe::init(void)>			1	0	1	0	100.00	100.00	100.00	100.00
Segment 1			1							
<TicTacToe::paint(void)>			9	0	9	0	100.00	100.00	100.00	100.00
Segment 1			83							
Segment 2			249							
Segment 3			83							
Segment 4			747							
Segment 5			249							

FIGURE 19 Xcover Example Output 1

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 > module name > segment > text of segment;

S1 expansion tree is: filename > module name > method invocation pair > text of method invocation 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 method invocation 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

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 for Java™.

6.1 Purpose

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

A method invocation pair (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 20 on page 53). 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 Xcalltree 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

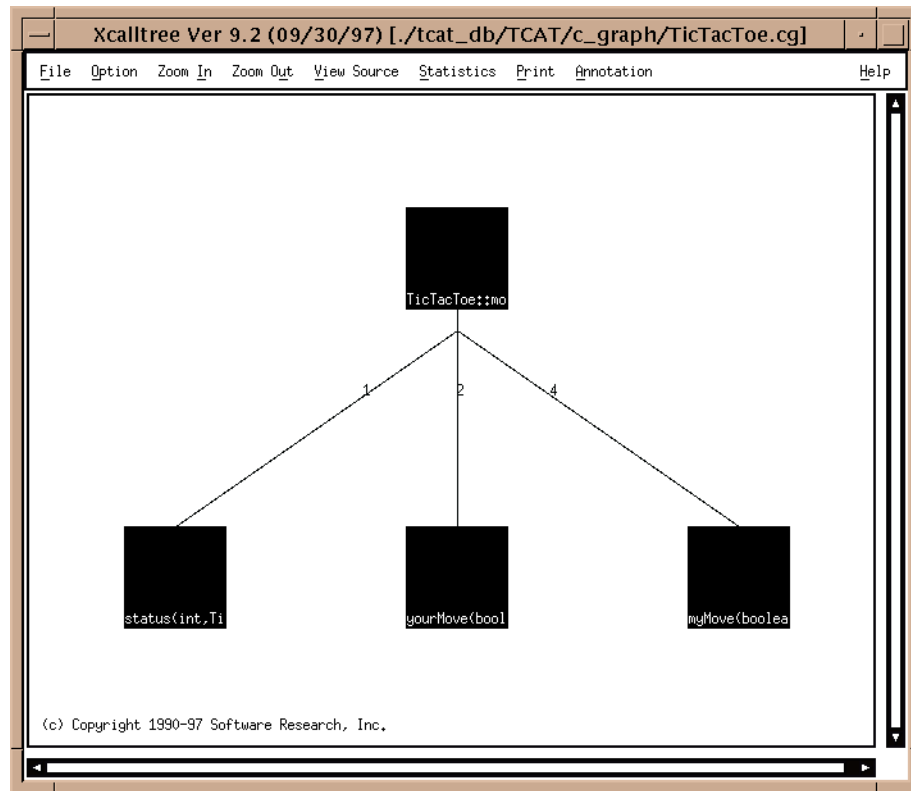


FIGURE 20 Xcalltree main window

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

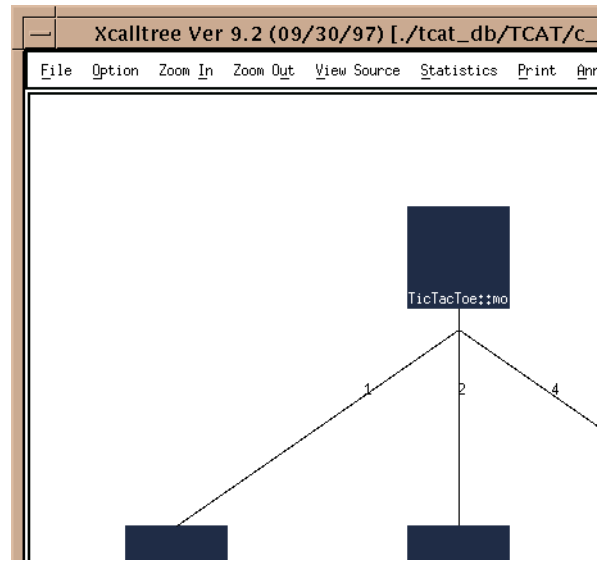


FIGURE 21 File Pull-Down Menu

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 22 on page 58 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

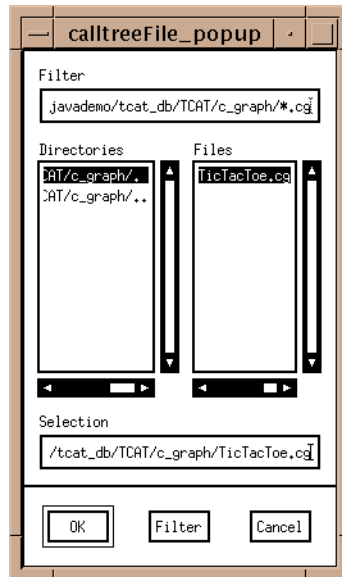


FIGURE 22 Calltree File Selection Dialog Box

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

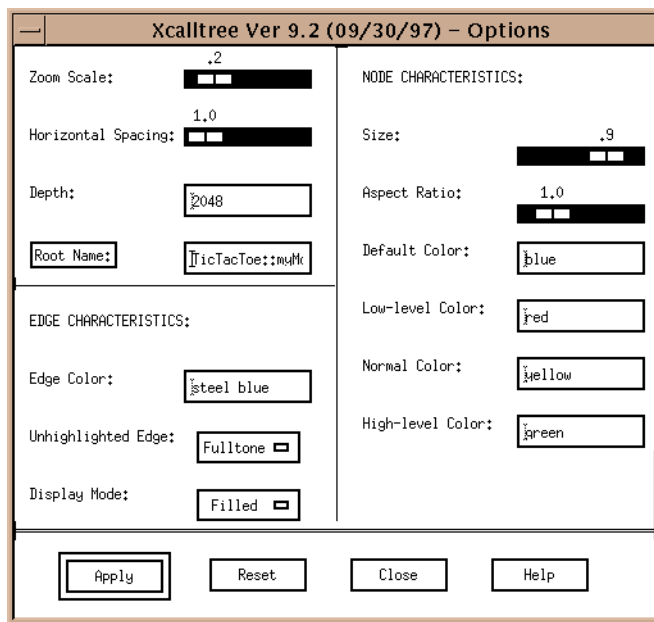


FIGURE 23 Option window

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

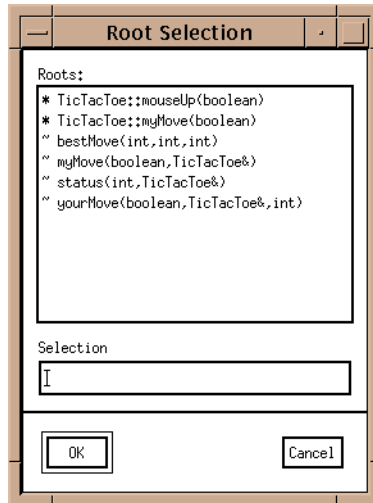


FIGURE 24 Root Name Selection window example 1

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 25 on page 63. 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 “*” in front of the name (as in Figure 26). They are sorted to 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.

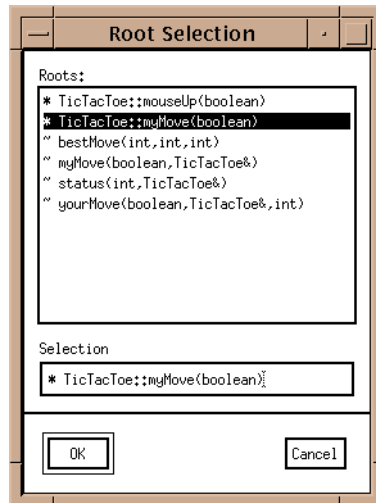


FIGURE 25 Root Name Selection Window Example 2

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

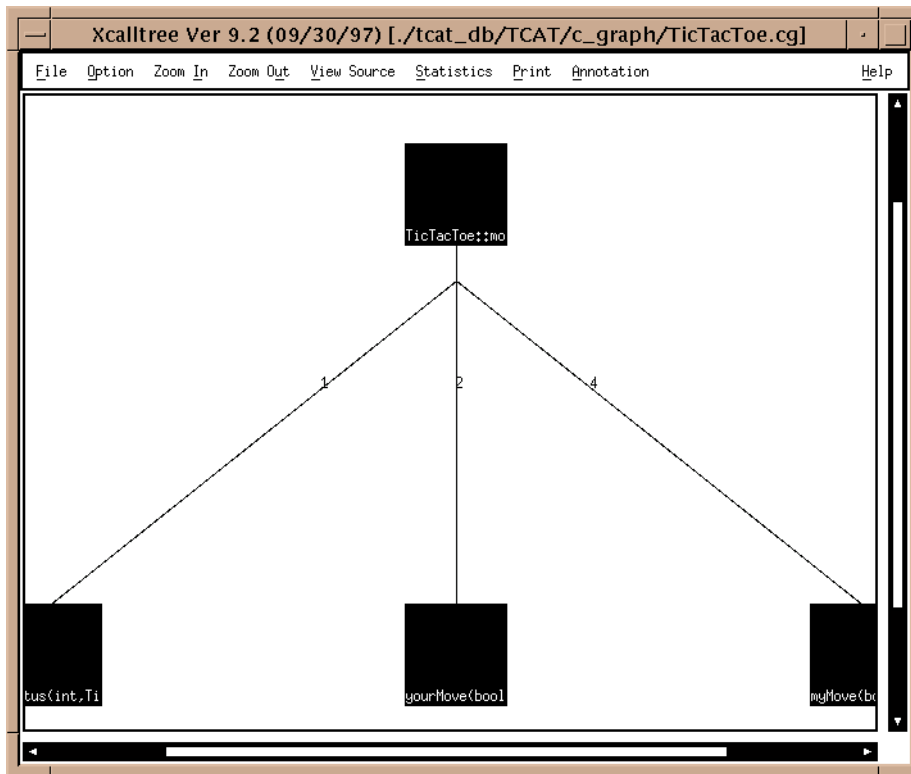


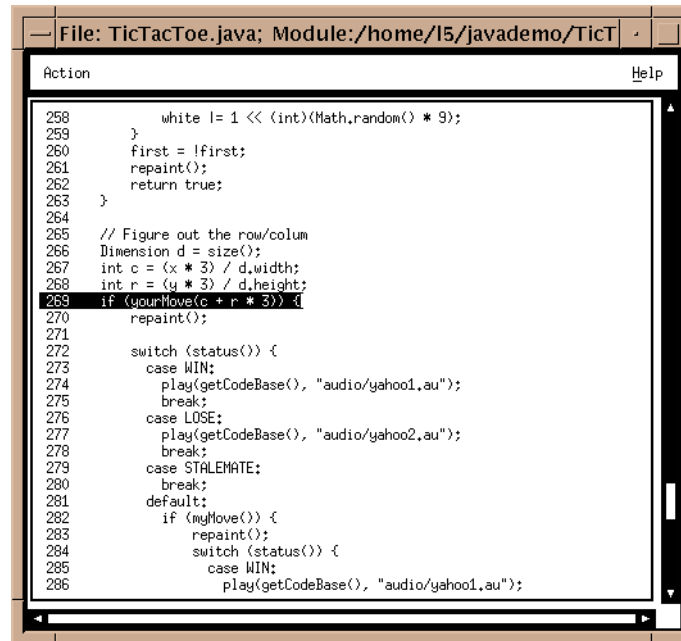
FIGURE 26 Zoom In Option illustrated

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 26 on page 66, after clicking on **Zoom In** once, and the same calltree, depicted in Figure 20 on page 53.

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: TicTacToe.java; Module:/home/l5/javademo/TicT
Action Help
258     while l= 1 << (int)(Math.random() * 9);
259     }
260     first = !first;
261     repaint();
262     return true;
263 }
264
265 // Figure out the row/column
266 Dimension d = size();
267 int c = (x * 3) / d.width;
268 int r = (y * 3) / d.height;
269 if (yourMove(c + r * 3)) {
270     repaint();
271 }
272 switch (status()) {
273     case WIN:
274         play(getCodeBase(), "audio/yahoo1.au");
275         break;
276     case LOSE:
277         play(getCodeBase(), "audio/yahoo2.au");
278         break;
279     case STALEMATE:
280         break;
281     default:
282         if (myMove()) {
283             repaint();
284             switch (status()) {
285                 case WIN:
286                     play(getCodeBase(), "audio/yahoo1.au");
```

FIGURE 27 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 method invocation pair (call pair) you clicked on (or the first method invocation pair (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

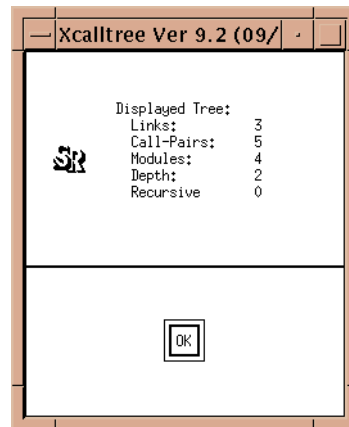


FIGURE 28 Statistics Window

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

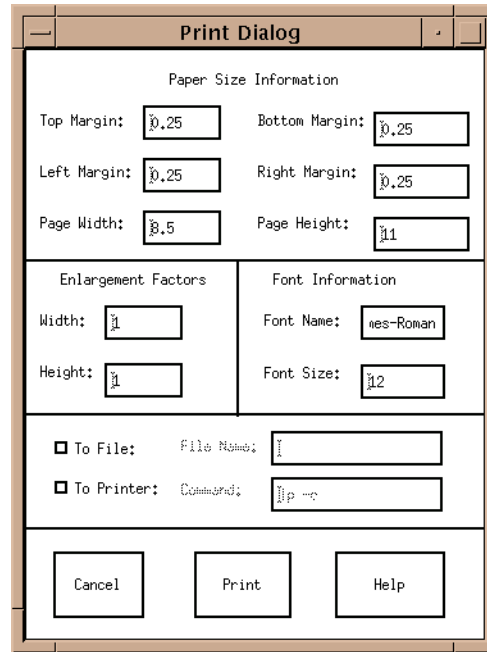


FIGURE 29 Print Window

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

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

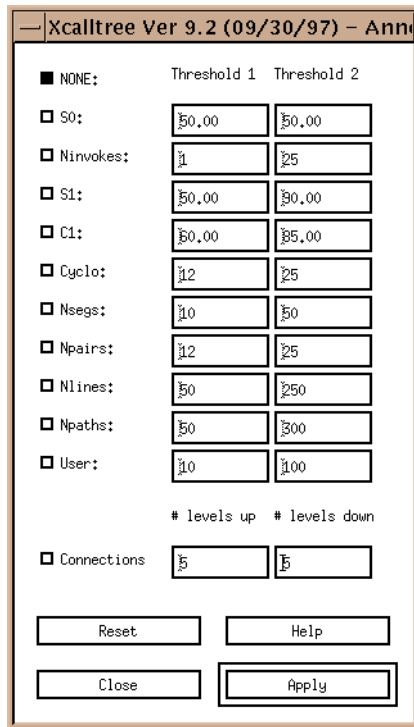


FIGURE 30 Annotation Window

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**
Method invocation pair (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 on the **Action** menu and select **Search**. This displays 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 Archive file. Shows only two colors on display, low and high. The Archive file must be from <i>TCAT for Java</i> . If not, silence.	50.00 / 50.00
Ninvokes	Number of times module was invoked, from Archive file. The Archive file must be from <i>TCAT for Java</i> . If not, silence.	1 / 25
S1	S1 value for module, from Archive file. Module name must appear in Archive ; else no default color. The Archive file must be from <i>TCAT for Java</i> . If not, silence.	50.00 / 90.00
C1	C1 value for module, from Archive file. Assumes module name appears in Archive ; else no color. The Archive file must be from <i>TCAT for Java</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 for Java</i> Ver 1.0.	10 / 50
Npairs	Number of call pairs in module, from tcat_db . The tcat_db must be from <i>TCAT</i> . If not, silence.	12 / 25
Nlines	Number of source lines. The number of non-blank lines in the module, from tcat_db . The tcat_db must be from <i>TCAT</i> . Requires use of <i>TCAT for Java</i> Ver 1.0.	50 / 250
Npaths	Number of paths in module retrieved.	50.00 / 300.00
User	"(-1,0,1) = Xcalltree.user N Lo Hi " for all $N =$ pair-number. The default supplied sample of Xcalltree.user chooses values at random.	10 / 100
Connections	Up and Down callers/callees from clicked function.	5 / 5

Xdigraph

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

7.1 Purpose

The **Xdigraph** utility draws digraphs based on archive files from *TCAT for Java*. 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 31). 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 draw a digraph based on the `filename` given. The available switches have the following values:

- A `archive file` *Archive file name.* Default is 'Archive'.
- B `filename` *Spine file.* 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` *Highlight specified file.* File called will come up in "highlight" mode; program searches for file with *.pth* extension.
- h This switch brings up **Xdigraph**'s help window.

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

```
Xdigraph
```

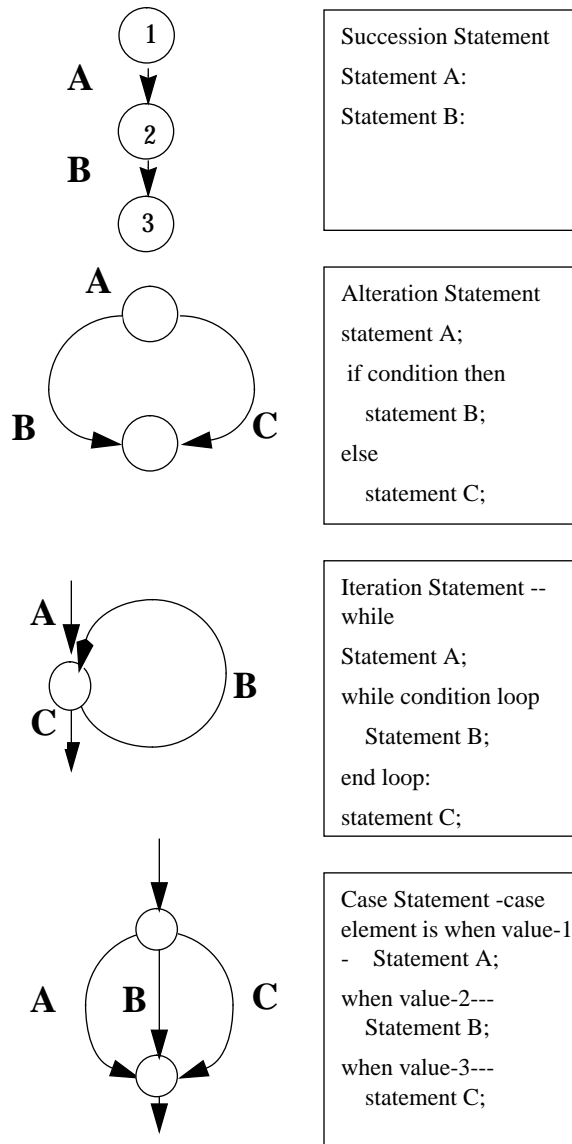



FIGURE 31 Program edges as represented in a digraph

7.4 Xdigraph Main Window

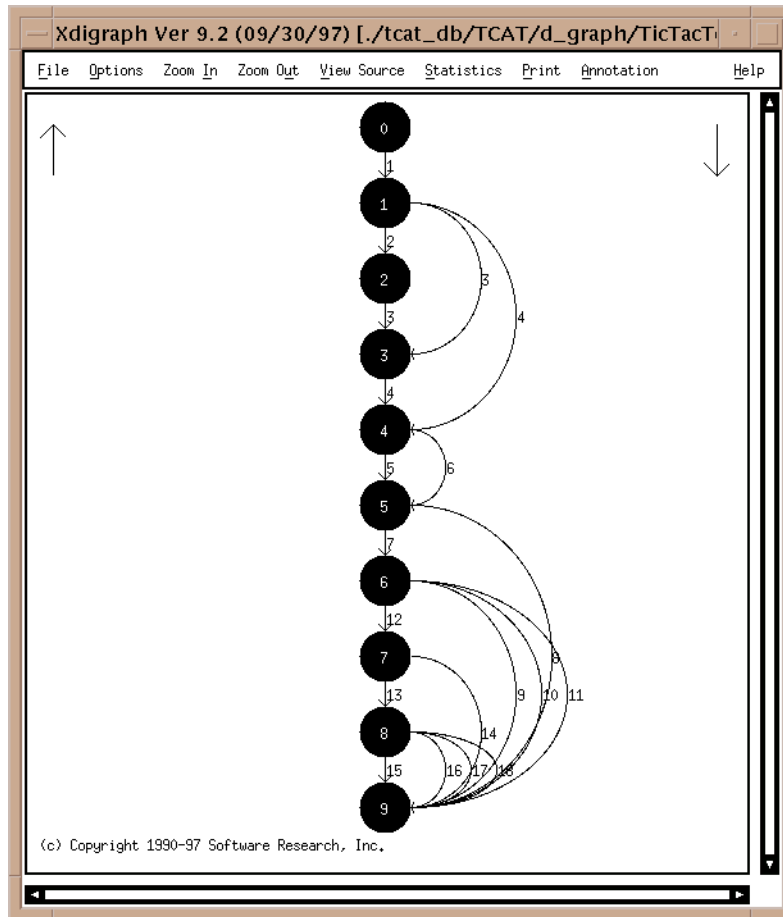


FIGURE 32 Xdigraph main window

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

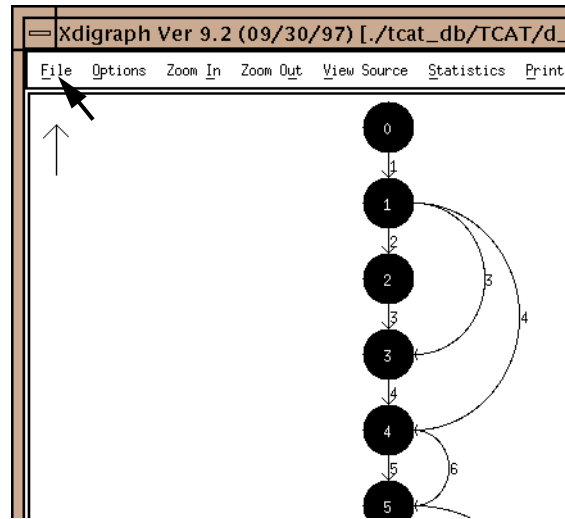


FIGURE 33 Digraph File Pull-Down Menu

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 34 on page 87) 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 34 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.

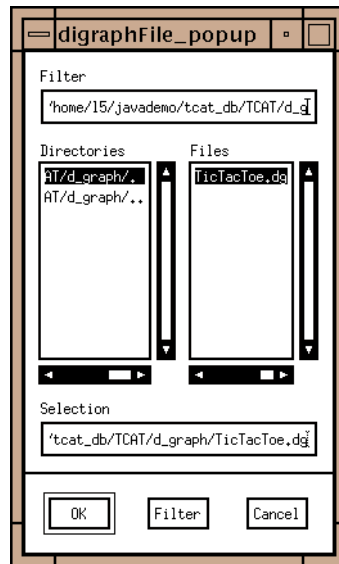


FIGURE 34 Digraph File Message Box

7.5.5.1 Filter

Limits the files that will be searched for to only those ending in **.dg.**

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

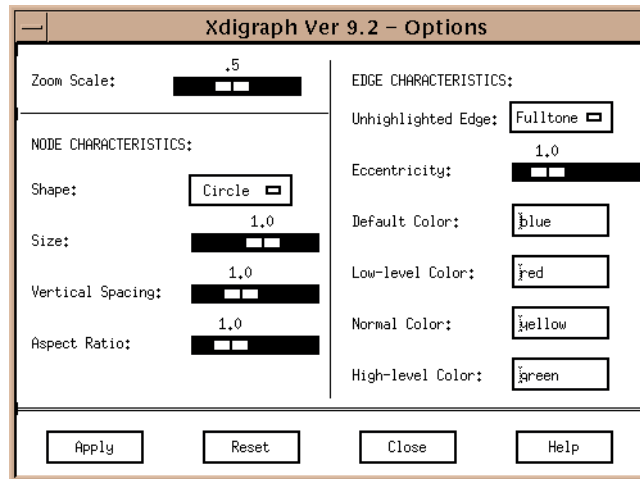


FIGURE 35 Xdigraph Options Window

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

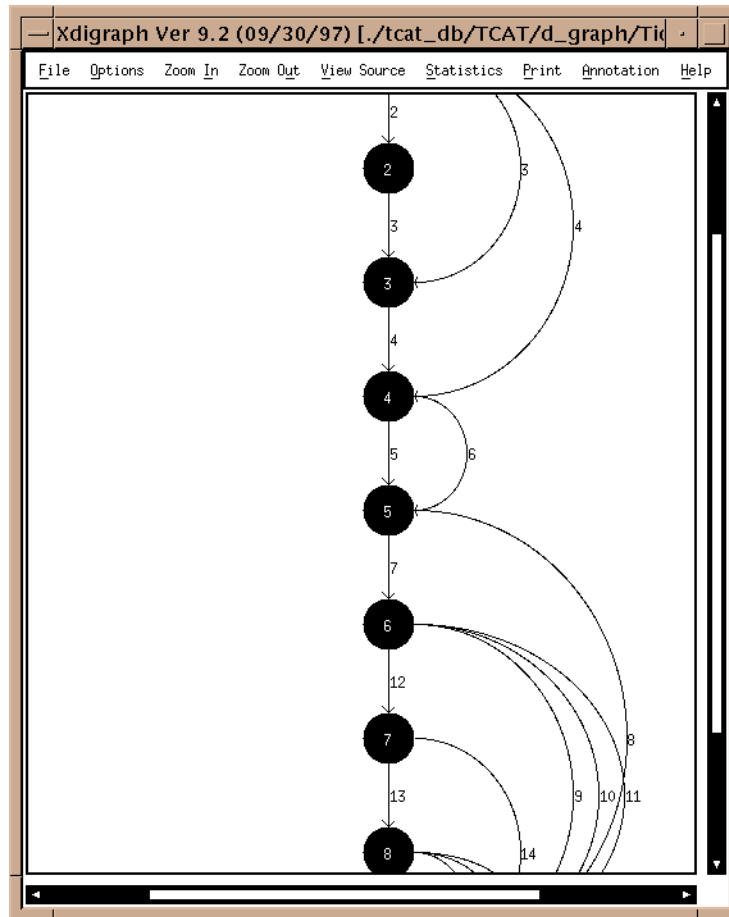


FIGURE 36 Zoom In feature illustrated

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 36, after clicking on **Zoom In** once, and the same digraph, depicted in Figure 32.

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

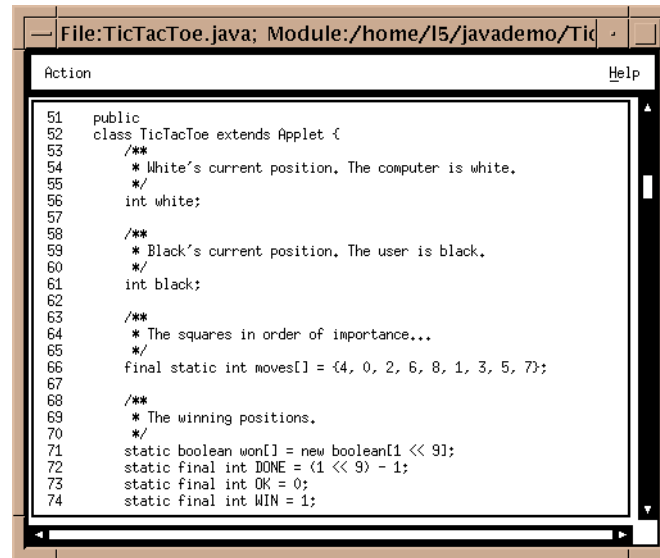


FIGURE 37 View Source Option Window

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

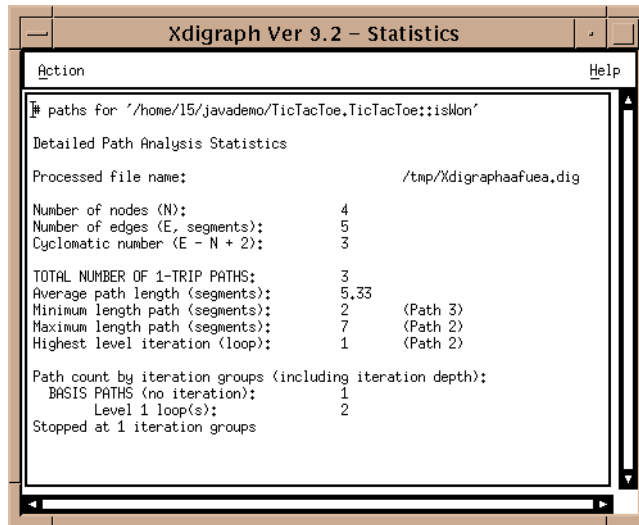


FIGURE 38 Statistics Option Window

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, but even so the calculation may show the “hour glass” waiting symbol.

When the limits are exceeded 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, calculated using the one-trip loop assumption.

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

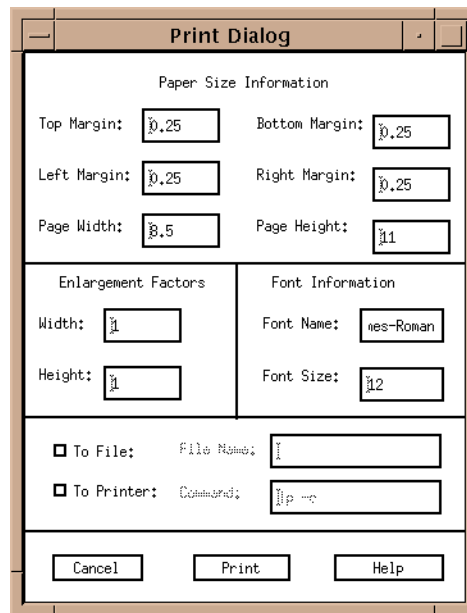


FIGURE 39 Print Dialog Window

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

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

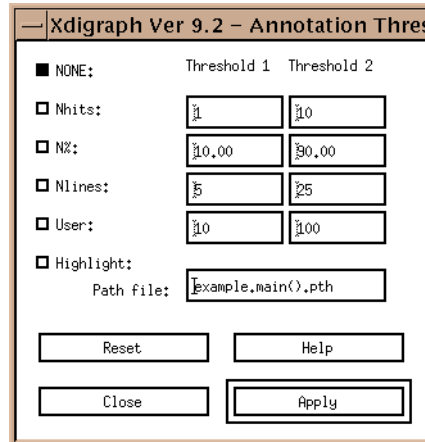


FIGURE 40 Annotation Thresholds Window

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 41, the annotation is for **User**.

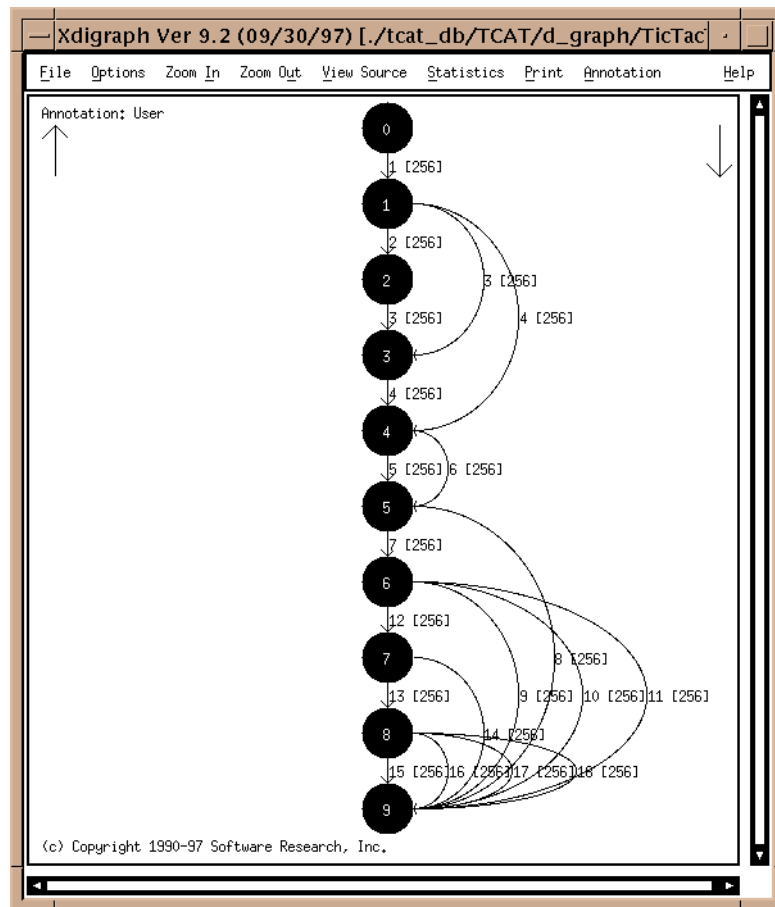


FIGURE 41 Sample Annotation for **User** Threshold

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 Archive file. The Archive file must be from <i>TCAT for Java</i> . If not, silence.	1 / 10
N%	Percent of total hits in module for this edge (segment), from local Archive file. The Archive file must be from <i>TCAT for Java</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 for Java</i> .	5/ 25
User	“-1,0,1) = Xdigraph.user <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</i> =1. (Path File) Left button moves up one path; right button moves down one path. If no path file is specified, apg will generate one.	N/A

TABLE 2

Quick Reference Guide for Xdigraph Annotations

Resource File Variables

This appendix describes the resource files supplied with TCAT for Java™, and applies to all editions of the product.

A.1 Overview

Within the **TCAT for Java** 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 for Java 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.

A.2 Variable Meaning

The variables within the resource file have the following meanings:

SRJ*config.option.instrumentS0: The state of the SO instrumentation button within the GUI.

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

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

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

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

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

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

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

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

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

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

SRJ*config.option.compilerCommand: The name of the compiler.

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

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

SRJ*config.option.instruCommand: The name of the instrumentor, ijava for "JAVA" applications.

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

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

The most important of these resources is the `SRJ*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.

A.3 Platform-Specific Examples

Below are the resource files for each supported “JAVA” platform.

A.3.1 Most Platforms

This is the “TCATjava.res” resource file for JAVA:

```
! Flags needed to instrument JAVA code

SRJ*config.option.instrumentS0:False
SRJ*config.option.runTarget:
SRJ*config.option.linkFlags:
SRJ*config.option.instrumentC++:True
SRJ*config.option.buildCommand:make -f
SRJ*config.option.linkCommand:cc -o
SRJ*config.option.instrumentAnsiC:False
SRJ*config.option.instrumentKRC:False
SRJ*config.option.runArgs:
SRJ*config.option.linkRuntimeModule:jrun.class
SRJ*config.option.linkLibraryFlags:-lm
SRJ*config.option.compilerCommand:javac
SRJ*config.option.buildFlags:
SRJ*config.option.instrumentC1:True
SRJ*config.option.instruCommand:ijava
SRJ*config.option.instrumentS1:False
SRJ*config.option.compilerFlags:
```

cover9 —TCAT for Java's Coverage Analyzer

This notes explains options for invoking and customizing the “cover9” coverage analyzer. This notes applies to all editions of TCAT C/C++ and TCAT for Java.

These are the options on how to invoke **cover9**. This command, used inside the **TCAT for Java** 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.

B.1 Command Line Invocation

The complete syntax for calls to **cover9** is listed below. Items enclosed in [brackets] are to be included zero or more times.

```
cover9 [tracefile [tracefile]]
      [-a old-archive]
      [-b file]
      [-c]
      [-C1]
      [-d name [name]]
      [-DI deinst-file]
      [-DL]
      [-f new-archive]
      [-h | -h name [name]]
      [-html | -html filename]
      [-H]
      [-N]
      [-n]
      [-nl namefile]
      [-NH]
      [-m]
      [-l | -l name]
      [-p]
```

[-P0]
[-P1]
[-q]
[-r report]
[-S0]
[-S1]
[-s]
[-SU]
[-T [threshold]]
[-w width]]

B.2 Cover9 Switch Definitions

The options may be used to vary the processing and reports generated by **cover9**. 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 **cover9** 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 *file* as a title for your reports. A maximum of 80 characters is allowed for titles.
- c** *Cumulative Report Switch.* This option prints the Cumulative report only.
- C1** *Branch Coverage Reporting Switch.* Turns on reporting of C1 or branch coverage.
Note: Unless at least one of **-C1**, **-S1**, or **-S0** is turned on, no coverage report will be generated.
- d name** *Module Name Delete Switch.* If this switch is present then the named modules, if found in the current execution, are deleted from the generated Archive file. Subsequently, **cover9** 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.
- DI deinst-file** *De-instrument Switch.* 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 *deinst-file* is to be excluded from coverage reporting. The module names can be specified in any format. White space (such as tabs, spaces) is ignored. *deinst-file* is also the file where new modules that pass the coverage threshold value (see the **-T** switch) will be written.
- DL** *De-instrument Module List Switch.* Allows the user to see which modules are excluded from coverage reporting. This switch is used along with the **-DI** switch. The list of excluded modules is printed at the end of the coverage report
- f new-archive** *New Archive File Name Switch.* 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 **cover9**, 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.

-h -h [name]	Linear Histogram Report Switch (-h).
-html [filename]	<i>HTML Switch.</i> If present, the current coverage report in html format will be generated. Normally the report is written to the file Coverage.htm (the default name), but you can rename the file with this switch. CAUTION: You will overwrite any file you name with this switch.
-l -l [name]	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, cover9 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.
-H	<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.
-m	<i>Minimal Output Switch.</i> When present, cover9 suppresses banner information, list of current options and trace file descriptions. The coverage report contains only the reports requested.
-N , -n	<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.
-NH	<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.

-nl <i>namefile</i>	<p><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.</p> <p>The names used must be specified one name per line. White space (tabs, spaces, etc.) on the line is ignored.</p> <p>The following reports are affected by the existence of a namefile:</p> <ul style="list-style-type: none">•Cumulative Report•Past Report•Not Hit Report•Hit Report•Newly Hit Report•Newly Missed Report. <p>The histogram outputs are not affected. There is a separate name mechanism that can be used to produce individual histogram reports.</p>
-NM	<p><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.</p>
-p	<p><i>Past Report Switch.</i> Print only the Past Test report; this option should be used in conjunction with the <i>-a</i> option when you want to analyze the overall performance of a set of past tests.</p>
-q	<p><i>Quiet Output Switch.</i> Suppress printout of current version and release information (this can be used to facilitate running cover9 in batch mode).</p>
-r <i>report</i>	<p><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.</p>
-S1	<p><i>Call-Pair Coverage Switch.</i> If present, the report will show call pair coverage.</p>

-S0 *Module Coverage Switch.* 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.

-s *Sort Switch.* This option produces output reports with module names sorted alphabetically.

-SU *Suppress Update Switch.* During processing, **cover9** will suppress updating of the archive file, either the default Archive or the file named by the **-f** switch. **cover9** will read the data in the archive file to form the basis for the “past test” information.

-T threshold *Coverage Threshold Switch.* 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 **-DI** *deinst-file* switch). If no threshold is specified, then the default value of 85 percent is assumed.

-w width *Report Width Switch.* Normally the reports generated by **cover9** 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 **cover9** 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 $128 - 21 = 107$. **WARNING:** Reports with high values for the **-w** option may contain long lines and may not be suitable for printing directly.

B.3 Error Processing

In case there is an error, **cover9** 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 74–78
archive files 45, 46, 49, 79
 overwriting 45

B

branch coverage, C1 2, 3, 31, 48, 75

C

C1 expansion tree 48
call pair coverage, S1 2, 3, 48, 75
caller-to-callee connections 69
calltree 4, 6
calltree display options 64–66
calltree graph
 basename.cg 15
cover utility 9
cover, coverage analyzer
 reports
 generating 21
 viewing 21–22
Cover9 Release Notes 111
cover9, coverage analyzer 111–116
 command line syntax 111
 invoking 111
cyclomatic complexity 75, 97

D

database reference file 15
directed graph 15

E

equivalence class coverage, Ct 2, 3, 6

F

font

 italics xv
 italix xv
font, bold face xv
font, courier xv

I

if 24
instrumenting an application 15, 16
instrumentor engine
 Java (ijava) versions 9, 31

J

jrunN.class utility 9

L

load setting 14, 29, 107

M

make files 38
manual organization xiv

O

object files 15, 38

P

PostScript (.ps) file 73
profile 15

R

regression testing 1
resource files 107–110
 definitions of variables 108
 most platforms 110
rm command 38
runtime object module
 linking 17
 run options 34

S

S1 expansion tree 48

Index

special text xv
static analysis 1

T

TCAT 12, 13
 building 38
 file filter 35–36
 instrument/compile options 30–31
 instrumenting 38
 instrumenting an application 38
 invoking 12–13, 25
 kill button 40
 link/build/run options 32–34
 main window 26–40
 saving settings 29
 selecting database name 29
 selecting utilities 27
 shell 27
 tutorial 11–24
tcat_db directory 15
TCAT-PATH 6
test coverage 1
text
 "double quotation marks" xv
 boldface xv
 italics xv
text, boldface xv
text, courier xv
text, italix xv
trace files 20, 45, 49
tutorial 11–24

V

vi editing program 38

X

X Window System 12
Xcalltree
 invoking 52
Xcalltree utility 4, 9, 38, 51–78
 annotating 74–78
 annotation 55, 57
 display options 54, 60–66
 displaying a calltree 57
 displaying statistics 54, 68–69
 file format 51
 file selection 54, 56–59
 help 55
 main window 53–57
 setting archive file 57
 print options 54, 70–73
 root selection 62–63
 viewing source code 54, 67
 zoom 54, 61, 66

Xcover utility 7, 9, 19, 23, 24, 38, 45–49
 invoking 46
 project files 49
 viewing source code 23
Xcoverj 46
Xdigraph utility 9, 38, 79–106
 annotation 102–106
 display options 89–93
 displaying source code 95
 displaying statistics 96–97
 file selection 85–88
 invoking 80
 main window 82–84
 print options 98–101
 zoom 90, 94
Xtcat utility 9
Xtcatj 12