Quick Start

Processing Scribble Using TCAT C/C++

TCAT C/C++ for Windows is a Test Coverage Analysis Tool that provides detailed tabular and graphical reports on tests of C and C++ software applications. It operates on Windows 95 and Windows NT 4.0.

Overview

This application uses Microsoft **Visual C++(MSVC)** example program **Scribble** to demonstrate how to create and view the coverage reports, calltrees and directed graphs of the trace files that **TCAT C/C++ for Windows** creates when an instrumented application is tested.



Figure 1 TCAT C/C++ Program Group

TCAT C/C++ for Windows



Figure 2 TCAT C/C++ Integrated with MS-Visual C++ v5.0 Main Window



Tool Bar

The options available from the Tool Bar are the frequently used TCAT C/ C++ for Windows features.



Figure 3 Tool Bar

Configure TCAT	Selects among modes of instrumentation.
Build Instrumented App.	Instruments an application.
Run Instrumented App.	Runs the instrumented application.
Analyze Cover	Analyze the coverage achieved from tests.
Run DiGraph	Digraph display for the selected object.
Run Calltree	CallTree display for the selected object.
Run SMARTS	Organizes and executes a collection of tests.
Run CAPBAK	Captures and plays back tool.

Scribble

Scribble employs many features of Microsoft Foundation Classes (MFC). There are several versions of **Scribble**, which become increasingly complex in each chapter. MSVC++ 5.0 has eight chapters; The present example uses Chapter 8. MSVC++6.0's **Scribble** example is in Chapter 7.



Figure 4Scribble, Chapter 8

This demonstration includes the following steps:

- **1.** Preparing the example application, Scribble, for instrumentation.
- 2. Instrumenting Scribble.
- **3.** Building an executable file, Scribble.exe.
- 4. Testing Scribble.
- 5. Displaying tabular and graphical reports on the test of Scribble.



Preparing and Instrumenting Scribble

There are two methods to instrument Scribble by either using options from the TCAT C/C++ Integrated with MS-Visual C++ v5.0 / v6.0 window or by using the TCAT C/C++ Program Group window.

Using the TCAT C/C++ Integrated with MS-Visual C++ v5.0 / v6.0 window

1. Select **File** | **Open Workspace**, then select the "**Scribble.dsw**" file from the TCAT-CPP\Examples\Example2\Scribble-vc5.0 directory.

Open Worksp	Jace		? ×
Look jn:	Scribble-VC5.0	- E	×
Hlp Bes			
Scribble.ds	w		
J			
File <u>n</u> ame:	Scribble.dsw		<u>O</u> pen
Files of <u>type</u> :	Workspaces (.dsw;.mdp)	•	Cancel



- **2.** From **Build** pull-down menu select Configurations, then click the **Add** button and type in "Coverage" as a new configuration name.
- 3. Select **Project** | **Settings**, then select **Win32 Coverage in the box of Setting For:**.

- 4. From **Project** select **Setting**.
 - Click on the **Scribble** project name, then click on the General tab menu, and type in "Coverage" to both the Output files and Intermediate files option.
 - Click on the C/C++ tab menu, then select the **Precompiled Head**ers, and select the **Not using precompiled headers** options.
 - Click on the "**stdafx.cpp**" file form **Scribble**, then select the **Pre-compiled Headers** and select **Not using precompiled headers** options.

Project Settings	? ×
Settings For: Win32 Release	General Debug C/C++ Link Resourc
Scribble Source Files Source Files Source Files Source Files mainfrm.cpp minfrm.cpp minfrm.cp	Category: Precompiled Headers <u>Reset</u> Not using precompiled headers Automatic use of precompiled headers Through header: Create precompiled header file (.pch) Through header: Use precompiled header file (.pch)
scribitm.cpp	Through header:
scribvw.cpp scrimac.hpj scrimac.r stdafx.cpp ⊕ ☐ Header Files	Project Options: /nologo /MD /W3 /GX /02 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_AFXDLL" /D "_MBCS" /Fo".\Release/" /Fd".\Release/" /FD /c
	OK Cancel

Figure 6 Project Setting Dialog Box



5. From Tools pull-down menu select Customize, then click on the Add-Ins AND Macro Files tab menu, and select SRCov Developer Studio Add-in option.



Figure 7Customize Option Dialog Box

- 6. Click on the **Configure TCAT Option** button.
 - Click on the **Instrumentor Options** tab menu, then select the **C1** and **S1** options.
 - Click on the **Runtime Selection** tab menu, then select the "**RUNTMDLL.lib**" (located in the Program directory) file.

Instrumenting Scribble

1. Click on the **Build Instrumented App** button.

The instrumented object files will be placed in the debug (or release directory if you choose) directory.

Executing the Instrumented Scribble

1. Click on the **Run Instrumented App** button, then test-drive the instrumented **Scribble** to create a trace file.

Using the TCAT C/C++ Program Group window

Setup using Microsoft Visual C++

In Microsoft Visual C++ v5.0 / v6.0:

- 1. Select **File** | **Open Workspace**, select **Scribble.dsw** (located in the TCAT-CPP\Examples\Example2\Scribble directory) as the project.
- 2. Select **Insert** | **Files into Project...** and add **RUNTMDLL.lib** (located in the Program directory) to the project.
- 3. Select Build | Build Scribble.exe.

In Microsoft Visual C++ v4.x:

- 1. Select **File** | **Open Workspace**, select **Scribble.mdp** (located in the Samples\Scribble directory) as the project.
- 2. Select **Insert** | **Files into Project...** and add **RUNTMDLL.lib** (located in the Program directory) to the project.
- 3. Select Build | Build Scribble.exe.

Instrument Using WinIC9

WinIC9 instruments the application under test in order to produce trace files of the test.

To instrument the example application:

1. Start up **WinIC9**.



Figure 8 WinIC9 Window

2. Select Scribble.cpp using the **Select** button. Note that more than one file can be selected and instrumented, and that instrumenting multiple files will result in a more thorough coverage report.

Select File(s)	to instrument		? ×
Look <u>i</u> n:	🔁 Scribble-VC5.0	• 🖻 🖸	* :::
Hlp Pri_Dir Release Res Childfrm.cp	Imainfrm.cpp Imainfrm.cpp	€‡ Stdafx.cpp	
, File <u>n</u> ame: Files of <u>t</u> ype:	"Scribdoc.cpp" "Scribble.cpp" CPP Files (*.cpp)	·	<u>O</u> pen Cancel

Figure 9 Select File(s) to Instrument

Note: More than one file can be selected and instrumented, and instrumenting multiple files results in more thorough coverage.

3. Select **Options** button.

Setting **Compiler Options** for the instrumenter. The TCAT instrumenter invokes the native compiler after completing its processing steps. To instrument a program correctly the compiler options need to be set correctly.

The compiler options very with your application and they can be copied directly from Visual C++ settings. To find the compiler options you need select **Setting** for the project. Then select the appropriate **Project Settings**. Select **C/C++**. The Option that are needed can be found in the field **Project Options**.

One example compiler options setting is listed below.

Scribble Debug Version compiler options:

/nologo /MDd /W3 /Gm /GX /Zi /Od /DWIN32 /D_DEBUG /D_WINDOWS / D_AFXDLL/D_MBCS/Fo". \Debug/"/Fd"./Debug/"/FD/c

Scribble Release Version compiler options:

/nologo /MD/W3/GX/O2/DWIN32 /NDEBUG/D_WINDOWS/D_AFXDLL/ D_MBCS/Fo".\Release/"/Fd"./Release/"/FD/c

- **4.** Select **Instrument**. A copyright box pops up before the instrumentation of each file. Click **OK** to proceed.
- **5.** During instrumentation, a command-line window displays messages and warnings. When instrumentation of a file is complete, a prompt appears. Type exit to proceed.
- 6. Select **Exit** from the **WinIC9** window.

The instrumentor has parsed the application's source code, looking for logical branches or segments and inserting markers (function calls).

Instrumenting **Scribble** will not change its functionality. When compiled, linked and executed, the instrumented application will behave as it normally does, except that it will write coverage data to a trace file. For more information on **TCAT C/C++ for Windows**' instrumentor, refer to Chapter 3 of the *Users Guide*.



Link Using Microsoft Visual C++

In Microsoft Visual C++:

1. Build Scribble.exe.



Figure 10 TCAT C/C++ Integrated with MS-Visual C++ v5.0 Main Window The preceding steps create an instrumented executable file for **Scribble**, which when executed will create a trace file.



Executing the Instrumented Scribble

- 1. Execute **Scribble** from **Microsoft Visual C++**.
- 2. Test-drive the instrumented **Scribble** to create a trace file.





3. Select **Exit** from the **Scribble File** menu.

The instrumented test produces a trace file, the trace file created by this "test," *Trace.trc*, resides in the *tcat_db* directory hierarchy in the Scribble directory, which in the next step **Cover** will use to produce coverage reports.



Viewing Coverage Reports with Cover

To view a coverage report of the trace file created by the execution of the instrumented version of **Scribble**:

- 1. Start up Cover.
- 2. From the File menu, select Open.
- **3.** In the **Open** dialogue, click on the filename Trace.trc from the tcat_db\Scribble directory created during instrumentation. The dialog box then asks for an archive file; ignore this request by clicking the **Cancel** button. A coverage report of the test of Scribble appears.

Cover displays trace and coverage information on your development project in a treelike list. You can click on a branch of the list to expand it and show its content, and also to contract it. The several fields in the report have the following meanings:

Hits The number of times the segment and call pair were executed during the test.

Count The number of segments and call pairs within the function.

C1 The percentage of branch coverage for each function.

S1 The percentage of call pair coverage for the function.

ace bo									
Project Name PL_Name Update Archive	Trace F Aachive P		Scabble/	/C4.2/PL	DV/kik_db/	PCNane	drace fro		
Current Authine	Hit: Fie	conde	Courts		C1 Daw	erage 1	St Cav	enope %	
Functione: 0 D	Segi	OP	Seal	CPI	Cu.	Cun	C#	Cin	
Project Totals :	11134	20	17	17	76.47	16.47	94.12	94.12	
CScribbleApp: DrAppAboutyoet1 Function Totals Segment 1 Calipair 1	2 2 (2)	2 2 2 22	1	1	108.00	100.00	100.08	180.00	ĺ
Function Totals Segment 1	200 200 2	000	1		108.00	100.00	100.00	180.00	ł
Function Totals Function Totals Segment 1	100 100 p	001 0	1		108.00	100.00	100.08	180.00	
Function Totals : Segment 1 Calipai 1	4 4 [4]	4 [4]	1	1	108.00	100.00	100.08	180.00	
Function Totals : Segment 1 CS withdolary: Indications of the	2 2 [2]	•	1		100.00	100.00	100.00	100.00	

Figure 12 Coverage Report on Scribble, with One Function Expanded to Show Segments



Viewing the Source Code Associated with Cover

You can view the source code associated with any segment numbers, or callpair numbers of the function in a coverage report by clicking on the segment numbers or callpair numbers. For example, double-click on a segment number. The code is displayed in a separate window with the calling statement highlighted in red.

CI COVER for Windows - Scribble	.cpp: Seg	ment ID	1					_	□×
<u>File View Window H</u> elp									
scribble_only.trc									
Project Name : Pri_Name	Trace F	File : C:	∖SR Testir	ng\Covera	ige\TCAT\E:	xamples\S	cribble-VC	5.0\Pri_Di	r\to
Update Archive	Archive F	File : N/	A						
Current Archive	Hits Re	ecords	Counts		C1 Cove	erage %	S1 Cov	erage %	
Functions: 8 0	Segs	CPs	Segs	CPs	Cur.	Cum.	Cur.	Cum.	
Project Totals :	3305	14	17	17	47.06	47.06	82.35	82.35	
Function Totals :		0 >×1	1	1	0.00	0.00	0.00	0.00	
Function Totals :		0 GMAP*)	1	0	0.00	0.00	0.00	0.00	
Function Totals :	np(CALA_MO 0 d CD ataEuck	O O	1	0	0.00	0.00	0.00	0.00	
Function Totals :	0,CD ataE xcr	nange") O	1	1	0.00	0.00	0.00	0.00	
EAboutDig::LAboutDig(void) Function Totals :	0	0	1	0	0.00	0.00	0.00	0.00	
EScribbleApp::InitInstance(int) Function Totals :	5	14	9	15	55.56	55.56	93.33	93.33	
CScribbleApp::CScribbleApp(void) Function Totals :	1	0	1	0	100.00	100.00	100.00	100.00	
CScribbleApp::GetMessageMap(C Function Totals :	AFX_MSGM 1833	AP*) 0	1	0	100.00	100.00	100.00	100.00	
Segment 1 CScribbleApp:: GetBaseMessage	1833 Map(CAEX	[<mark>1833]</mark> MSGMAP [®]	ŋ						
Function Totals :	1466	0	<u>́1</u>	0	100.00	100.00	100.00	100.00	
Cribble.cpp: Segment	ID 1								
<pre>BEGIN_MESSAGE_MAP(CScribbleApp, CWinApp)</pre>									
ON COMMAND	TD FTLE	PRINT	SETH	P. CMir	Ann::On	FilePr	intSet	սու	
For Help, press F1								NUM	

Figure 13 Source Code Displayed from Coverage Report



Viewing Directed Graphs with DiGraph

To view a directed graph (digraph) of the application:

- 1. Open up DiGraph.
- 2. Using the File menu, select Open.
- **3.** You are prompted for the name of the directed graph to view. Find the Scribble.dg file under the d_graph directory.
- **4.** The next prompt asks for the name of the database file. Select the Scribble.mdf file in the tcat_db\Scribble directory.

Step 1/3 - 0	pen DG File			? ×
Look jn:	🔁 d_graph	•	£	ở ∷ ∭
SCRIBBLI	E.dg C.dg			
File <u>n</u> ame:	×.dg			<u>O</u> pen
Files of type:	Digraph Files (*.dg)		•	Cancel

Figure 14 WinDiGraph Open Dialog Box

5. A window pops up listing the available functions (Figure 15). For this example, select **CScribbleDoc::DeleteContents[void]**.

Select a MDF ID	×
CScribbleDoc::~CScribbleDoc(void,int) CScribbleDoc::OnNewDocument(int) CScribbleDoc::Serialize(void,CArchive&) CScribbleDoc::AssertValid(void) CScribbleDoc::Dump(void,CDumpContext&) CScribbleDoc::OnOpenDocument(int,Cchar*) CScribbleDoc::OnOpenDocument(int,Cchar*) CScribbleDoc::InitDocument(void) CScribbleDoc::InitDocument(void)	OK Cancel

Figure 15 Select MDF ID Box



A directed graph depicting possible program flows of the function **CScribbleDoc::DeleteContents[void]** appears.



Figure 16 Directed Graph of Scribble



By clicking near the number associated with an edge and selecting the **View Source** button, you can call up and view the associated source code.



Figure 17 Viewing Associated Source Code from Digraph

The source code associated with Segment 2 appears in a new window. In this figure, the windows showing the digraph and the source code have been tiled.

Viewing a Calltree

- **1.** Start up CallTree.
- 2. Using the File menu, select Open.
- **3.** You are prompted for the name of the calltree to view. Find Scribble.cg under the c_graph directory.
- 4. You are prompted for the name of the database file. Find the Scribble.mdf file under the tcat_db directory.
- **5.** A **Select Function** list box appears. Select the **CScribbleDoc::Delete-Contents[void]** function.

Select a MDF ID	×
CS cribbleD oc:: ~CS cribbleD oc(void,int) CS cribbleD oc:: OnN ewD ocument(int) CS cribbleD oc:: Serialize(void,CArchive&) CS cribbleD oc:: AssertValid(void) CS cribbleD oc:: Dump(void,CD umpContext&) CS cribbleD oc:: DeleteContents(void) CS cribbleD oc:: DeleteContents(void) CS cribbleD oc:: InitD ocument(void)	OK Cancel

Figure 18 Select MDF ID Box

The following calltree depicting the selected function appears.







Viewing the Directed Graph Associated With a Calltree Node

For each node in your calltree, you can easily display an associated directed graph. Select the Root node of the calltree. Notice that the **View Digraph** button on the toolbar now has a red arrow, indicating that it is available. Click this button. You will see a directed graph of the **CScrib-bleDoc::DeleteContents[void]** function.



Figure 20 DiGraph Displayed from Calltree Window

Viewing the Source Code Associated With a Calltree

You can view the source code associated with any node in a calltree by clicking on the corresponding edge. For example, click on the edge running from the root node to the left-most node. Once the edge is selected, it will be displayed thicker. Notice that the **Source Code** button on the Tool Bar has a red arrow. Click on this button to display the associated source code. The code is displayed in a separate window, with the calling statement highlighted in red.

🚜 Calltree for Windows - Scribdoc.cpp:
<u>File View Options Window Help</u>
SCRIBDOC - CScribbleDoc::DeleteContents(void)
C Scribble Doc::Delete
C Soribble Doc::On Edit CTyped PtrList::Remo C Ole Server Doc::Dele
Get Embedded Item CSoribble Item::Copy
💽 Scribdoc.cpp:
void CScribbleDoc::DeleteContents()
{ while (!m_strokeList.IsEmpty()) {
delete m_strokeList.RemoveHead();
) COleServerDoc::DeleteContents();
} void CScribbleDoc::InitDocument() {
m_bThickPen = FALSE;
<pre>m_nThinWidth = 2; // default thin pen is </pre>
For Help, press F1

Figure 21 Source Code Displayed from Calltree



TCAT C/C++ for Windows: Analysis of Reports

In the following analysis, a coverage report shows that a certain function, **CScribbleDoc::DeleteContents[void]**, has been tested 75.00%.

CI COVER for Windows - trace.trc Eile View Window Help									
E S R									_ 🗆
Project Name : Pri_Name Update Archive /	Trace Fil Archive Fil	e: [C:\S e: [N/A	icribble-V	C4.2\Prį	_Dir\tcat_db\	Pri_Name	\trace.trc		
Current Archive	Hits Rec	ords	Counts		C1 Cove	rage %	S1 Cov	erage %	
Functions : 41 0	Segs	CPs	Segs	CPs	Cur.	Cum.	Cur.	Cum.	
Project Totals :	274789	79999	75	64	76.00	76.00	87.50	87.50	
CScribbleDoc::NewStroke(CStroke*) Function Totals : Segment 1 Segment 2 Callesir 1	13 13 [13] 0 [0]	26	2	2	50.00	50.00	100.00	100.00	•
Calipair 2 Calipair 2 CScribbleDoc::InitDocument(void) Function Totals : Segment 1 Calipair 1	3 3 [3]	13 [13] 3 3 [3]	1	1	100.00	100.00	100.00	100.00	
CScribbleDoc::DeleteContents(void) Function Totals : Segment 1 Segment 2 Segment 3 Segment 4 Callpair 1 Callpair 2 Callpair 3 CScribbleDoc::OnOpenDocument(int,f	31 7 [7] 17 [17] 7 [7] 0 [0] Cchar*)	48 24 [24] 17 [17] 7 [7]	4	3	75.00	75.00	100.00	100.00	
For Help, press F1								NUM	

Figure 22 Coverage Report Showing C1 Coverage of 75.00% on the Function CScribbleDoc::DeleteContents[void]

The function consists of one segments and one callpairs. This coverage report shows that segments 1 and 3 were hit seven times each, segment 2 was hit 17 times, and segment 4 not once. The callpair 1 was exercised 24 times, callpair 2 was exercised 17 times, and callpair 3 was exercised seven times. The following few pages show graphical views of these numerical results.

In Figure 16, TCAT C/C++ for Windows graphs CScribbleDoc::Delete-Contents[void] and its relations. The calltree shows the callpairs in CScribbleDoc::DeleteContents[void], and the digraph shows possible program flows through CScribbleDoc::DeleteContents[void] divided into segments.





Note that the calltree shows three callpairs. These callpairs are shown in the coverage report in Figure 21, which have been exercised 24, 17, 7 times respectively. The coverage report shows that the percentage of S1 coverage (coverage of call pairs) was 100% for this function.

Note that the digraph shows three segments. The coverage report in Figure 21 shows that the test of **Scribble** hit three of four segments . The coverage report shows that the percentage of C1 coverage (branch coverage) was 75.00%.



To look at source code associated with callpairs, highlight the graphic lines connecting the functions shown in the calltree.



Figure 24 Calltree and Source Code Associated with One Callpair

TCAT C/C++ for Windows

To look more closely at the segments, highlight one of the graphic lines in the digraph by clicking on it close to the number. Then use the Source Code button to display the associated source code.



Figure 25 Digraph and Source Code Associated with One of Its Segments

Closing TCAT C/C++ for Windows

To close TCAT C/C++ for Windows:

- Select File|Exit from the menu bar of each open program, or
- Double-click on the frame window **Close Box** of each program.

You have now seen all of TCAT C/C++ for Windows' main features.

