



CodeTEST® Software Analysis Tools for Freescale™ StarCore™ MSC8100 Family Getting Started Guide

924-75525
Rev B
April 2006





FreescalTM and the Freescale logo are trademarks of Freescale Semiconductor, Inc. CodeTEST and CodeWarrior are trademarks or registered trademarks of Freescale Semiconductor, Inc. in the United States and/or other countries. All other product or service names are the property of their respective owners.

Freescale Semiconductor's CodeTEST product is protected under U.S. Patent 5,748,848.

Copyright © 2005-2006 by Freescale Semiconductor, Inc. All rights reserved.

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

How to Contact Us

Corporate Headquarters	Freescal Semiconductor, Inc. 7700 West Parmer Lane Austin, TX 78729 U.S.A.
World Wide Web	http://www.freescale.com/codewarrior
Technical Support	http://www.freescale.com/support

Table of Contents

1	Introduction	5
	About the Documentation	5
	Software Manuals	5
	Hardware Manual	6
	Online Help	6
	What is CodeTEST for StarCore?	6
	The Process for Monitoring Applications with CodeTEST Tools	9
	How Do You Set Up CodeTEST Tools?	10
2	Setting up the Tools	11
	Before You Begin	11
	What You Will Do in this Chapter	12
	Preparing Your Environment	13
	Validating the Probe Connection	14
	Setting up Data Collection	19
	Setting Up for Instrumentation in the CodeWarrior IDE	19
	Collecting Data	24
	Viewing Data	29
	At this Point	32
	What to Do Next	32
A	CodeTEST Plugin for CodeWarrior IDE Reference	39
	Before You Begin	39
	Configuring a Project for Instrumentation	40
	Targets	40
	CodeTEST File Mappings Panel	40
	CodeTEST Instrumentation Settings Panel	42
	Troubleshooting	44
B	Instrumenting for Memory Analysis	45
	Index	53



Table of Contents

Introduction

This manual shows how to get started with CodeTEST® Software Analysis Tools for a Freescale™ StarCore™ DSP target. It contains the following sections:

- [“Setting up the Tools” on page 11](#)
- [“CodeTEST Plugin for CodeWarrior IDE Reference” on page 39](#)

This chapter covers concepts that will help you understand how to set up applications for the CodeTEST Tools. It contains the following sections:

- [“About the Documentation” on page 5](#)
- [“What is CodeTEST for StarCore?” on page 6](#)
- [“The Process for Monitoring Applications with CodeTEST Tools” on page 9](#)
- [“How Do You Set Up CodeTEST Tools?” on page 10](#)

About the Documentation

NOTE For important information available after publication of the document set, please refer to the product Release Notes and README file included with your CodeTEST software.

In addition to this document, you should be familiar with the related materials in the CodeTEST document set, which are provided in PDF format in the installation `doc` directory.

Software Manuals

- *CodeTEST Software Analysis Tools Quick Start Guide*: CodeTEST software installation procedures and licensing information.
- *CodeTEST Software Analysis Tools User’s Guide*: how to use the CodeTEST Manager to capture and display data.
- *CodeTEST Instrumenter Reference Manual*: Instrumenter components, switches, and supported functions.
- *CodeTEST Software Analysis Tools Scripting Guide*: how to write CodeTEST scripts. Also includes the *CodeTEST Manager API*.

Introduction

What is CodeTEST for StarCore?

Hardware Manual

- *Setup and Installation Guide for the CodeTEST Probe*: installation and configuration procedures for the CodeTEST Hardware Probe.

Online Help

- CodeTEST Online Help: context-sensitive, quick reference, and procedural information.

What is CodeTEST for StarCore?

CodeTEST Tools for embedded developers and test engineers provide visibility into an application as it runs on a target.

You can use the CodeTEST Tools to trace code execution and measure performance and coverage. For more information about using the CodeTEST Tools, see the *CodeTEST Tools User's Guide*.

The Trace Tool — displays software execution history in three synchronized views.

- The **Trace** view displays branch points, function entries and exits.
- The **Execution History** view graphically displays the relationship between task context and function calls.
- The **Source Code** view displays the source code.

The Performance Tool — tracks function entries and exits as your application executes.

- The **Performance** view displays the amount of CPU time consumed by each function.
- The **Call Pair** view dynamically identifies the relationships between the program functions.
- The **A/B Timers** view displays the timing relationship between two selected events.

The Coverage Tool — tracks which statements of each function actually execute.

CodeTEST Tools measure the following three coverage levels.

- Statement Coverage (SC)
- Decision Coverage (DC)
- Modified Condition/Decision Coverage (MC/DC)

The Memory Tool — displays memory activity in two views.

- The **Memory** view tracks your program's dynamic allocation and deallocation of memory.
- The **Memory Errors** view displays memory allocation errors.

If you are using CodeTEST for StarCore with an RTOS, see also the *Getting Started Guide* for that RTOS.

See “CodeTEST Tools Capability Specifications” in the *CodeTEST Tools User’s Guide* for additional information.

NOTE Source view coverage color highlighting may be incorrect due to incompatibilities with the preprocessor. The instrumentation process and analysis of coverage data is correct.

CodeTEST Probe

CodeTEST Tools require a Probe to collect data from an embedded application as it runs on a target.

The Hardware Probe sends the data to the CodeTEST Manager on a workstation. The Manager generates data views that can be used to:

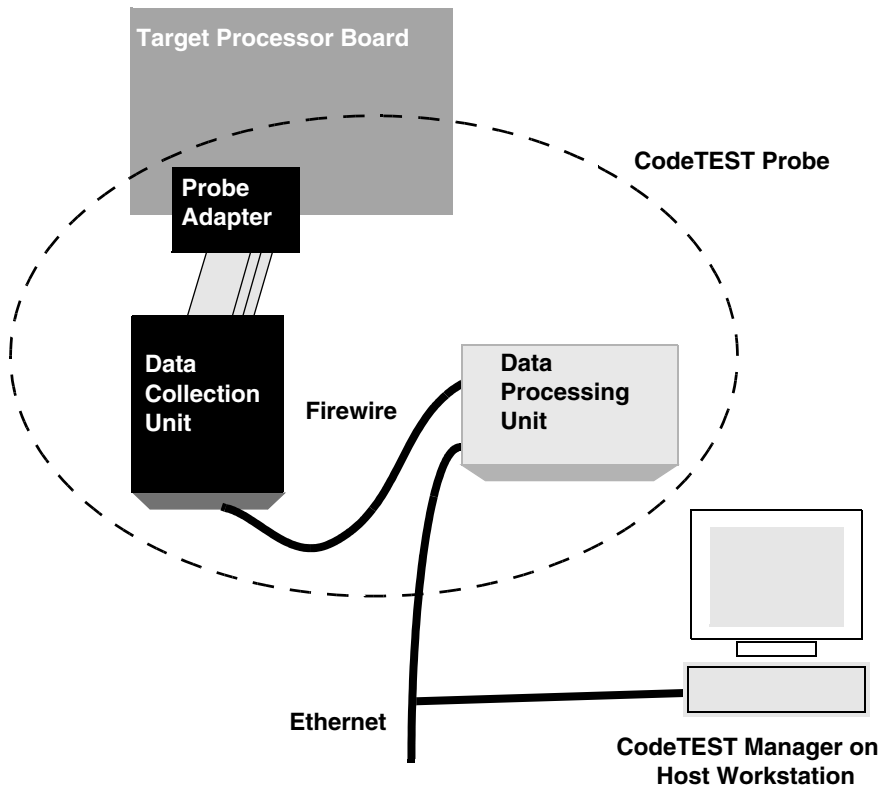
- Analyze performance
- Analyze coverage
- Trace code execution

[Figure 1.1](#) shows a typical system using a CodeTEST Hardware Probe.

Introduction

What is CodeTEST for StarCore?

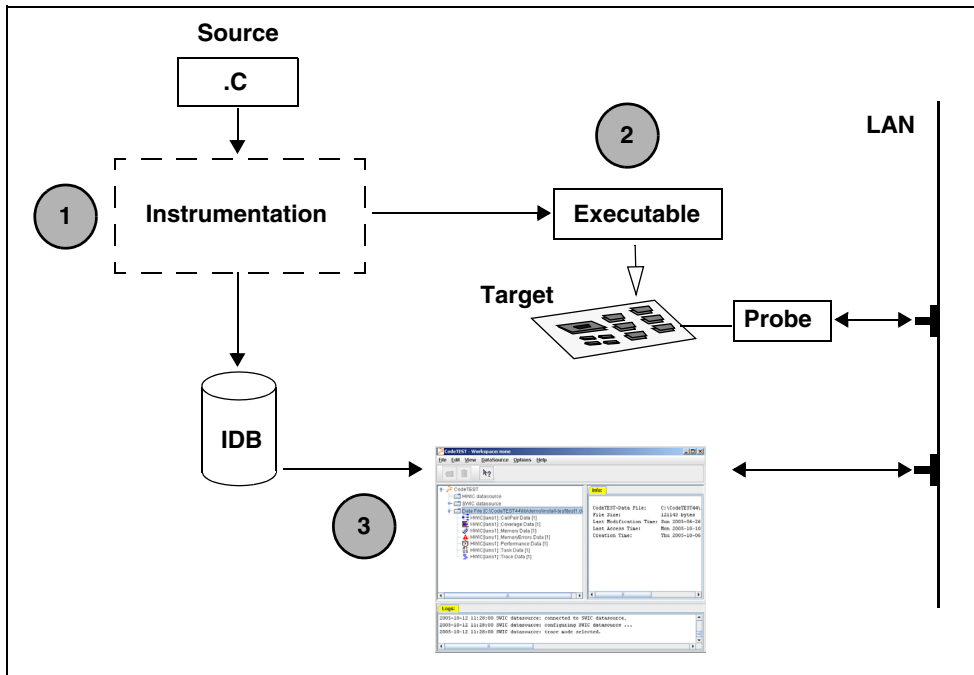
Figure 1.1 Typical Hardware Probe Installation



The Process for Monitoring Applications with CodeTEST Tools

The process that you will use to monitor your application with CodeTEST Tools is shown below.

Figure 1.2 Monitoring Applications with CodeTEST Tools



- 1 You will modify your environment to insert a CodeTEST utility called the Instrumenter into the preprocessing phase of your compiler. When you compile your code, the Instrumenter inserts test point instructions, called tags, into the executable. It also creates an Instrumentation Database (IDB) file that CodeTEST Tools use to resolve this tag data with source code. You use the CodeWarrior™ Development Studio for StarCore DSP IDE to set up instrumentation.
- 2 After you build your executable and download it to the target, you can start collecting data. As the executable runs on the target, it sends data to the CodeTEST Probe.
- 3 You can use the CodeTEST Manager to control data collection from the Probe and to view the data, trace code execution, and analyze coverage, performance, and memory.

Introduction

How Do You Set Up CodeTEST Tools?

How Do You Set Up CodeTEST Tools?

Chapter 2 provides instructions that show how to use a demo application that we provide.

We will show you how to:

- Set up the environment.
- Connect the CodeTEST Manager to the Probe and run a simple application called `taskwalk` to verify the Probe connection.
- Instrument a sample application called `CT_DEMO` using the CodeWarrior IDE, and set up the CodeTEST Manager to collect and view coverage, trace, and performance data.

Appendix B provides instructions on setting up an application to collect memory data.

NOTE Be sure to read [“Before You Begin” on page 11](#) to make sure that your environment is set up for CodeTEST Tools.

Setting up the Tools

This chapter shows how to prepare your environment, validate your Probe connection, and instrument and monitor a sample application called CT_DEMO. The instructions show how the CodeTEST Tools and the CodeWarrior™ Development Tools are used together to provide application visibility.

This chapter has the following sections:

- [“Before You Begin” on page 11](#)
- [“What You Will Do in this Chapter” on page 12](#)
- [“Preparing Your Environment” on page 13](#)
- [“Validating the Probe Connection” on page 14](#)
- [“Setting up Data Collection” on page 19](#)
- [“Viewing Data” on page 29](#)
- [“What to Do Next” on page 32](#)

For instructions that show how to use more advanced options, see the *CodeTEST Instrumenter Reference Manual* and the *CodeTEST Tools User’s Guide*.

Before You Begin

Before you begin, make sure that:

Your development environment has...	You know how to...
Freescalar CodeWarrior StarCore compiler. Application code can be successfully built (compiled and linked). A project configured to not include pre-compiled headers.	Build your application and modify the compiler definition and build sequence.
A connection (e.g., JTAG) established with the target processor that is used to run the application code.	Load images to the target processor.
A target processor that boots correctly and can run application code independent of CodeTEST Tools.	Start applications on the target processor.

Setting up the Tools

What You Will Do in this Chapter

These steps have been performed...	Who does this?
The “core” CodeTEST Tools v4.2 are installed on the workstation or network and a license file is installed on a license server.	A system administrator or someone who can obtain IP addresses, install the license server, and resolve network issues. Reference: <i>CodeTEST Tools Quick Start Guide</i>
The CodeWarrior Development Studio for StarCore v2.6 is installed.	A system administrator or software engineer. Reference: CodeWarrior Tools documentation
The CodeTEST Probe is connected to the workstation through a TCP/IP network. The Probe has an IP address and responds to a ping command from the workstation.	A system administrator or software engineer. Reference: <i>Setup and Installation Guide for the CodeTEST Probe</i>
The CodeTEST Probe is connected to the target board. The location for the CodeTEST tag port is identified in the target memory map.	A hardware engineer or someone who has knowledge of the target, including the memory map, schematics, pinouts, signal timing, and peripheral setup. Reference: <i>Installation Instructions: Nexus Adapters</i>

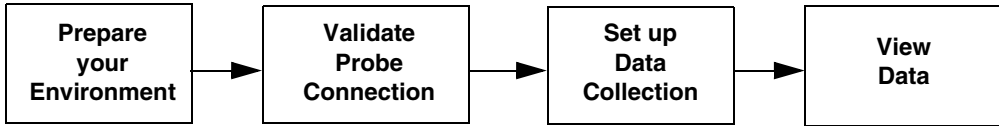
You know or have...	Where do you find this?
Your target processor type (e.g., StarCore).	A hardware engineer.
The compiler you are using (e.g., CodeWarrior).	A build engineer.
The network protocol you are using (TCP/IP).	A hardware or software engineer.
The directory where the core CodeTEST product was installed. (This is <i>CT_HOME</i> .)	A system administrator.

What You Will Do in this Chapter

This chapter shows how to validate your Probe connection and set up CodeTEST Tools to monitor a sample application called *CT_DEMO*. The process is shown below. This basic process applies to any environment, regardless of its level of complexity.

NOTE We will be using the CodeWarrior StarCore stationery to set up sample projects. Additional information is provided for your existing projects.

Figure 2.1 Setup Steps



- [Preparing Your Environment](#)
Copy the required files to your CodeWarrior installation and make sure that your environment variables are set correctly.
- [Validating the Probe Connection](#)
Use the CodeTEST Manager to configure the Probe and connect to the Probe. Then build, load, and run the `taskwalk` application to verify that the CodeTEST Manager can receive data from the Probe.
- [Setting up Data Collection](#)
Instrument the `CT_DEMO` application. When you compile the application, the CodeTEST Instrumenter inserts test point instructions (“tags”) in the code. It also creates an instrumentation database (IDB) file that maps these tags to the code. Set up the data source in the CodeTEST Manager for data collection.
- [Viewing Data](#)
Verify the data collection results in the Trace, Coverage, Memory, and Performance views.

Preparing Your Environment

Make sure that you have installed the core CodeTEST Tools before performing the additional steps here. See the *CodeTEST Tools Quick Start Guide* for instructions.

NOTE Please read the Release Notes for information regarding installation requirements for your operating system.

To install the CodeTEST Instrumenter plugin files

1. In the CodeTEST installation, locate the file `starcore.zip` in the CodeTEST installation directory `%CT_HOME%\lib\cw-plugins`.
2. `starcore.zip` consists of file: `starcore-CodeWarrior.zip`. Unzip the contents of `starcore-CodeWarrior.zip` to the top of your CodeWarrior for

Setting up the Tools

Validating the Probe Connection

StarCore installation directory, using the folder names specified in the zip file. These files add support for CodeTEST instrumentation to the CodeWarrior IDE.

To set up the environment

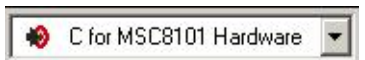
1. Set the `CT_TARGET` environment variable to `mwerks-starcore`. `CT_TARGET` is used by the CodeTEST Tools to identify your compiler and CPU.
2. Add the path to the CodeWarrior command line tools to your path:
`CodeWarrior_Install_Dir\StarCore_Support\compiler\bin`

Validating the Probe Connection

To validate the Probe connection and configuration, build the **uninstrumented** taskwalk application, configure the Probe, and monitor taskwalk as shown in the following instructions.

To build taskwalk

1. Start the CodeWarrior IDE for StarCore.
2. Select **File > New**.
3. Select **StarCore Stationery**. Enter taskwalk for **Project name**, and click **OK**.
4. In the **New Project** dialog box, select **MSC8101ADS**, and click **OK**.
If you are not using the MSC8101ADS variant, select the stationery for your variant, and click **OK**.
5. When the project opens, in the target field, select **C for MSC8101 Hardware**




6. Add the file `%CT_HOME%\lib\utils\taskwalk.c` to the project by right-clicking on the **code** directory on the **Files** tab, selecting **Add Files**, and browsing to the file. In the **Add Files** dialog box that appears, be sure the **C for MSC8101 Hardware** target is selected, and click **OK**.
7. Remove the default `main.c` file from the project by right-clicking on the filename in the **code** directory on the **Files** tab and selecting **Remove**.

8. In the file `%CT_HOME%\lib\utils\taskwalk.c`, add the following lines in the `#define` section at top of the file:

```
#define TAG_DEST 2
#define CT_PORT 0x21000000
```

This definition of the CodeTEST tag port (`CT_PORT`) typically works for a CodeWarrior build of `taskwalk`. You can change this value to the tag port address for your system, however the address must be writable by target code and must not interfere with other code symbols.

NOTE For guidance on selecting the port address, see the *Setup and Installation Guide for the CodeTEST Probe* and the CodeTEST application note for the processor.

9. Save and close `taskwalk.c`.
10. Click the **Make** button  to build `taskwalk`. You will run `taskwalk` after configuring the Probe in the next section.

To configure the Probe

1. Start the CodeTEST Manager.
 - From the **Start** menu or run `%CT_HOME%\bin\ctmgr`.

The Manager window has three “panes” that we will refer to in the rest of this section:

 - The **Info** pane provides configuration information about the data.
 - The **Logs** pane shows error, warning, and debug messages.
 - The **Workspace** pane is used to manage *data sources* and open or close *data sets*.

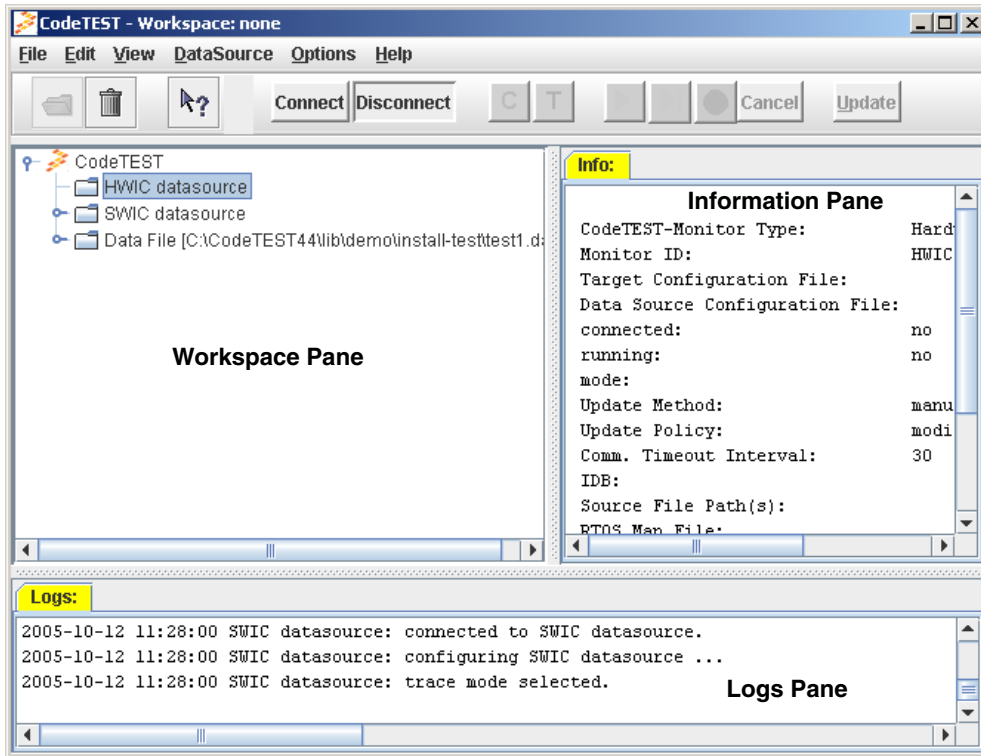
For information about the CodeTEST Manager, see the *CodeTEST Tools User's Guide* and online Help.

NOTE A *data source* refers to any CodeTEST monitor or source of data (e.g., a Probe is a data source). *Data sets* are the result of gathering data from a data source.



Setting up the Tools




Validating the Probe Connection

Figure 2.2 CodeTEST Manager Window



When a data source is highlighted, the toolbar at the top of the window is used to control data collection.

Button	Description
Connect	Establishes the connection to the Hardware Probe data source.
Disconnect	Disconnects the CodeTEST Manager from the Hardware Probe. Disconnecting does not shut down the Probe. It can continue to gather data from instrumented applications.
	Places the data source in Continuous mode. Continuous mode allows the data source to track the instrumented application performance and coverage performance.
	Puts the data source in Trace mode. Trace mode allows you to track the instrumented application(s) detailed execution until the trace buffer is filled.

Button	Description
 or Alt-s	Starts the data source and collect data.
 or Alt-x	Stops the data source.
 or Alt-c	Continues operation from stopped state, without clearing continuous mode data.
Cancel	Stops trace data collection, without uploading the trace buffer. Stopping, whether automatically (because the trace buffer is full), or manually (using the Stop button), causes the Manager to upload and process the entire trace buffer. If you want to stop execution trace and do not want to upload and process the trace buffer, use the Cancel button.
Update or Alt-u	Uploads continuous mode data on demand. If update is used while the data source is collecting data, then the resulting data sets may be inconsistent. For consistent data sets, stop the data source before updating. This button is disabled if the update method is set to automatic.

- From the **DataSource** menu, select Hardware Probe to open the **Config & Control** window **DataSource** tab.

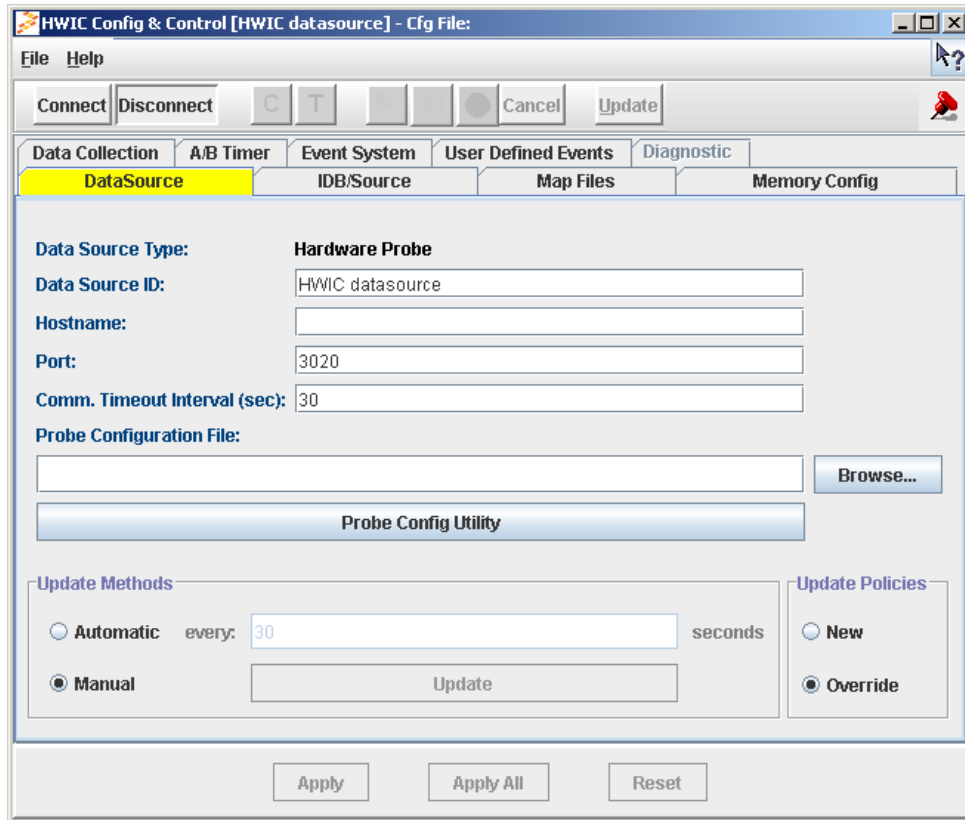
The fields within the tabs configure the Manager. For more about the **Config & Control** window, see [“Step 2: Set up the Manager for your application” on page 32](#), the Online Help, and the *CodeTEST Tools User’s Guide*.

- Type a name for the data source in the **Data Source ID** field.
- Type the host name or IP address of your Probe in the **Hostname** field.
- Under **Update Methods**, select **Automatic** to load data from the Probe every 30 seconds.
- Click **Apply** to apply the settings.

Setting up the Tools



Validating the Probe Connection

Figure 2.3 HWIC Config & Control Window





7. Click **Probe Config Utility** to open the **Probe Utility** window.
8. From the **Probe** menu, select **Universal** and fill in the fields on the **Probe Utility** window. See the CodeTEST application note for the MSC8101ADS for guidance on required settings.
9. Click **Apply** to save the Probe configuration file and apply it to the Probe.
10. On the **Config & Control** window, click **Apply All**.
11. On the Manager **File** menu, select **Save > Workspace File As** and save the workspace as `ct_demo.ctwsp`.

To monitor taskwalk

1. Click the CodeTEST Manager **Connect** button to connect to the CodeTEST Probe.
2. Click the CodeWarrior IDE **Run** button  to download and start the taskwalk application.
3. Click the **Diagnostic** tab on the CodeTEST Manager **Config & Control** window.
4. Enter `taskwalk.data` in the **Output File** field and click **Start** . Diagnostic data collection and analysis will begin.

NOTE The CodeTEST Probe collects data as the taskwalk application runs on your target. The collected data is compared to a known good file and the results are displayed in the **Diagnostic** tab.

5. Observe the results.
 - If the green check  is highlighted, your Probe is connected and configured correctly.
 - If the red X  is highlighted, there is a problem with your connection or configuration or taskwalk is not running. Verify the hardware connections and the configuration settings (you may need to change the frequency or phase shift). If you cannot find the problem, contact Customer Support.
6. Reboot your target to return it to a known, clean state.

Setting up Data Collection

Follow the procedures in this section to instrument a demonstration application (CT_DEMO) using the CodeWarrior IDE. Then set up the CodeTEST Manager for data collection.

Setting Up for Instrumentation in the CodeWarrior IDE

The general configuration steps listed below for instrumenting a project are expanded in this section.

1. Open an existing project or create a new project.

Setting up the Tools

Setting up Data Collection

2. Configure a target for instrumentation.
3. Map source file extensions to the CodeTEST Instrumenter and instrumented file extensions to the CodeWarrior Compiler.
4. Specify Instrumenter options.
5. Build the project.

To create a CT_DEMO project

TIP For your application, either open the project for which you want to add instrumentation, or create a new one appropriate for your system.

1. Start the CodeWarrior IDE for StarCore.
2. Select **File > New**.
3. Select **StarCore Stationery**, enter `taskwalk` for **Project name**, and click **OK**.
4. In the **New Project** dialog box, select **MSC8101ADS**, and click **OK**.

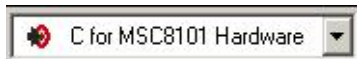
If you are not using the MSC8101ADS variant, select the stationery for your variant, and click **OK**.

To configure a target for instrumentation

TIP You may want to have two targets for your application - one that is set up for instrumentation and one that is not. To create a CodeTEST target, click on the project **Targets** tab, and from the **Project** menu select **Create Target**, enter a name, and click **OK**.

Add CT_DEMO source files to the project and configure the CodeTEST target for your hardware:

1. When the project opens, in the project target field, select **C for MSC8101 Hardware**



2. Add the three `.c` files from `%CT_HOME%\cttarget\dcu\demo\ct_demo\src` to the project by right-clicking on the **code** directory on the **File** tab, selecting **Add Files**, and browsing to the files. In the **Add Files** dialog box that appears, be sure the **C for MSC8101 Hardware** target is selected, and click **OK**.

3. In the file `main.c`, add a declaration for the `CT_DEMO` entry point after the include lines:


```
extern void AppMain(void);
```

4. Rewrite the `main()` function as follows:

```
int main(void)
{
    AppMain();
    return 0;
}
```


NOTE For `CT_DEMO`, the CodeTEST tag port format used is an address, and is specified to the Instrumenter by `-tags-to-address=addr` in the **CodeTEST StarCore Options** panel in a later step. You will need to decide on a tag format appropriate for your system. If you choose to use the default format, `-tags-to-port`, use of the `ct_port[0]` array is assumed and you need to define a `ct_port` symbol in your linker command file, as described in [“To define the tag port” on page 46](#).

5. Save and close `main.c`.
6. If you want to debug the application, be sure that debug mode is selected for the `taskwalk.c` file on the project **Files** tab. A black dot will be displayed in the

column  when you click the space in the column under the icon.

To map source file extensions for CodeTEST instrumentation

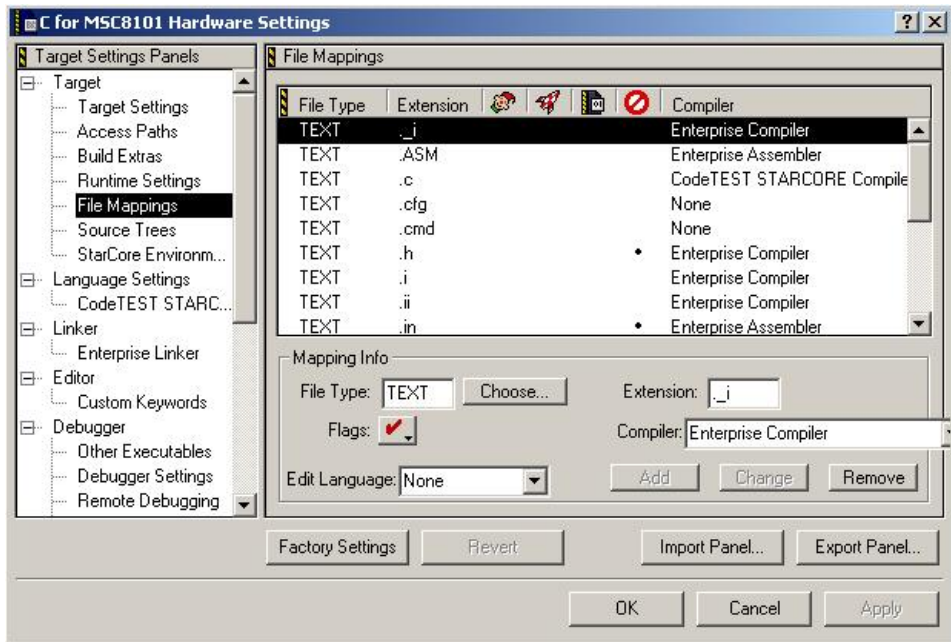
Associate your source file settings with the CodeTEST instrumentation process.

1. Select the CodeTEST target and click the project settings icon .
2. Select **Target > File Mappings**.
3. For each source file type (e.g., `.c`) that you want to instrument, select **CodeTEST STARCORE Compiler** and click **Change**.
4. Define a new file type:
 - File Type: Text
 - Extension: `._i`

Setting up the Tools

Setting up Data Collection

- Compiler: Enterprise Compiler
- Click **Add**.
5. Click **Apply** to apply the settings.



To specify CodeTEST instrumentation options

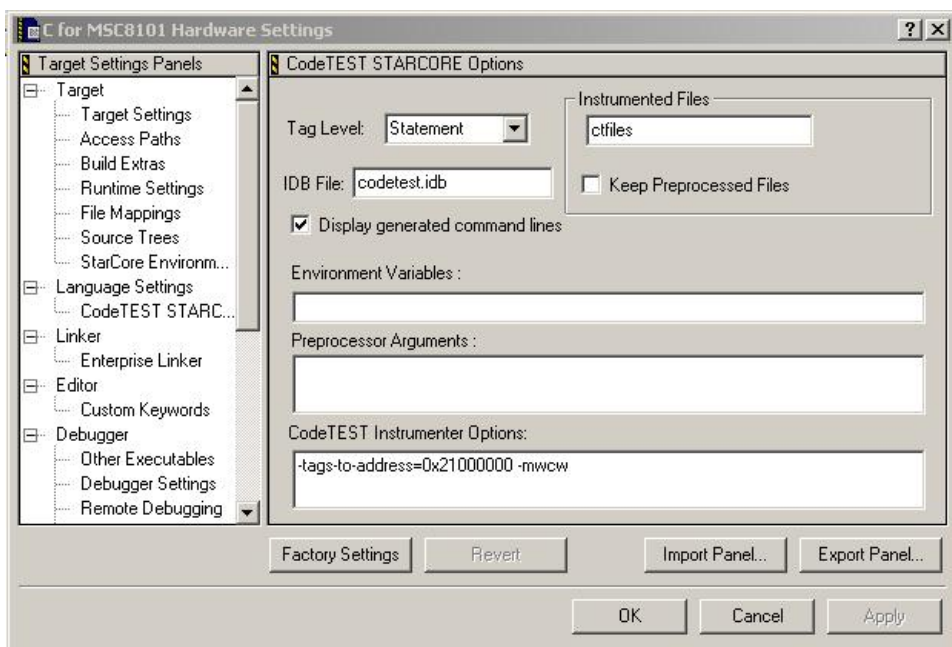
Specify the instrumentation options for the project in the **Target Settings Panel** for CodeTEST instrumentation. For extensive information about instrumentation options, see the [“CodeTEST Instrumentation Settings Panel” on page 42](#) and the *CodeTEST Instrumenter Reference Manual*.

1. In the **Target Settings Panel**, select **Language Settings > CodeTEST STARCORE Options**.
2. Verify the options are set as follows:
 - **Tag Level:** Statement
 - **IDB File:** codetest.idb
 - Check **Display generated command lines**.

- **CodeTEST Instrumenter Options:** `-tags-to-address=0x21000000`
`-mwcw`

The option `-tags-to-address=0x21000000` sets the value of the CodeTEST tag port to 0x21000000, which typically works for a CodeWarrior build of `ct_demo`. You can change this value to the tag port address for your system, however the address must be writable by target code and must not interfere with other code symbols. See “The CodeTEST Tag Format” in the *CodeTEST Instrumenter Reference Manual* for more information.

The option `-mwcw` specifies the StarCore Enterprise compiler to the CodeTEST Instrumenter.



3. Click **OK** to apply the settings and close the window.

NOTE Instrumenting for memory analysis is not covered in this chapter; see for instructions.

Setting up the Tools

Setting up Data Collection


NOTE When you set up instrumentation for your application, see [“CodeTEST Plugin for CodeWarrior IDE Reference” on page 39](#) for additional information about the options panel.

To build the project

NOTE If you previously imported your project with a different version of the CodeTEST plugin, “Unable to load include paths” error messages may be reported. Delete the project files (e.g., *proj_name_emul.mcp*) and directory (e.g., *proj_name_Data*), re-import the project, and repeat the steps in the instrumentation section.

Click the **Make** button  to build the CT_DEMO project.

NOTE All CodeTEST Instrumenter messages will display in the **Errors & Warnings** window.

NOTE After you build the application, the instrumented files will be added to the project. If you want to debug your instrumented source code, be sure that debug mode is selected for each instrumented file on the project **Files** tab and build again. A black dot will be displayed in the column  when you click the space in the column under the icon.

Collecting Data

Use the CodeTEST Manager to control data collection and view the application data.

To set up the Manager

1. If you closed the Manager after validating the Probe, open it, select **File > Recent Workspaces**, and open the *ct_demo.ctwsp* workspace that you saved when you configured the Probe.
2. In the workspace, select the data source for your Probe and right-click on it to open a context menu.
3. From the context menu, select **Configuration** to open the **Config & Control** window.

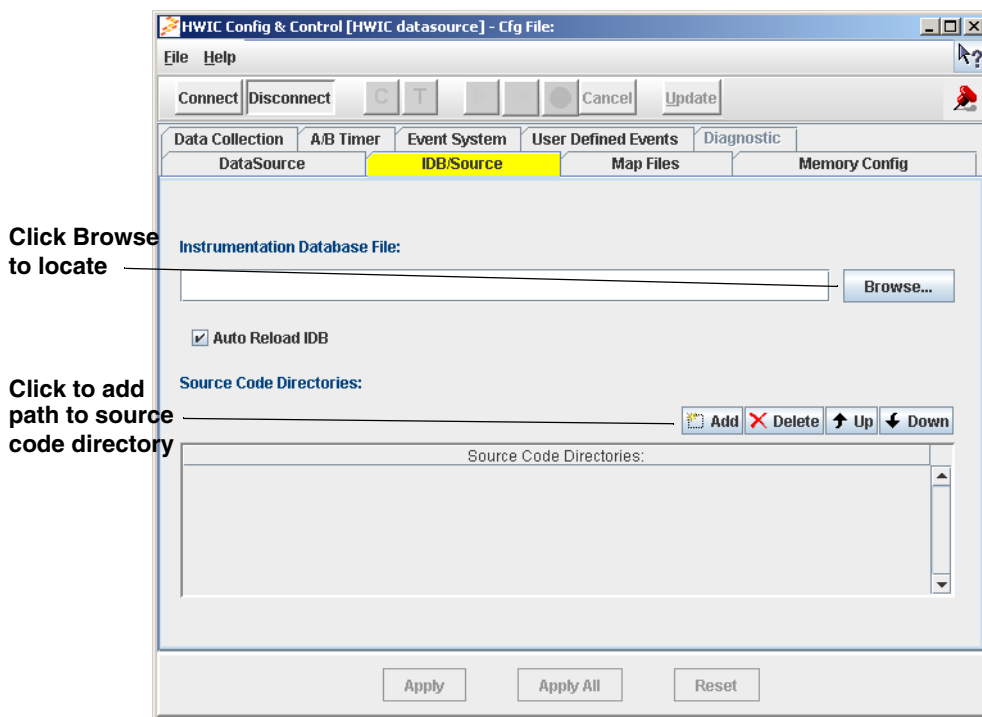
The fields within the tabs configure the Manager. For more information on the **Config & Control** tabs, see [“Step 2: Set up the Manager for your application” on page 32](#), the Online Help, and the *CodeTEST Tools User’s Guide*.

For CT_DEMO, we will perform the minimum configuration and use many default settings. When you configure the Manager for your application, you may want to change these default settings.

To specify the IDB and source files

1. In the **Config & Control** window, click the **IDB/Source** tab.

Figure 2.4 IDB/Source Tab





2. Browse to the `codetest.idb` file that was created when you instrumented the CT_DEMO application:

`Project_directory\output\codetest.idb`

Setting up the Tools

Setting up Data Collection

For more information on IDB files, see the section “The CodeTEST Instrumenters” in the *CodeTEST Instrumenter Reference Manual*.

3. Select the file and click **Open**.
4. In the Source Code Directories group, click the **Add** button  to add a new directory path.
5. Double-click the highlighted box that appears, and click the ... button . Browse to the directory path for the application source code:

```
%CT_HOME%\cttarget\dcu\demo\ct_demo\src
```
6. Click **Open** to select the path.

This pane allows you to specify any directories that contain source code for the instrumented program, so that you can browse to that code from the data windows. The `src` directory is all that is needed for `CT_DEMO`.
7. Click **Apply**.

To set the data set options

1. Click the **Data Collection** tab.
2. Under Data Type, select the **Coverage** and **Performance** options.
3. Under Coverage Level, select **SC** and **Block** (Default).

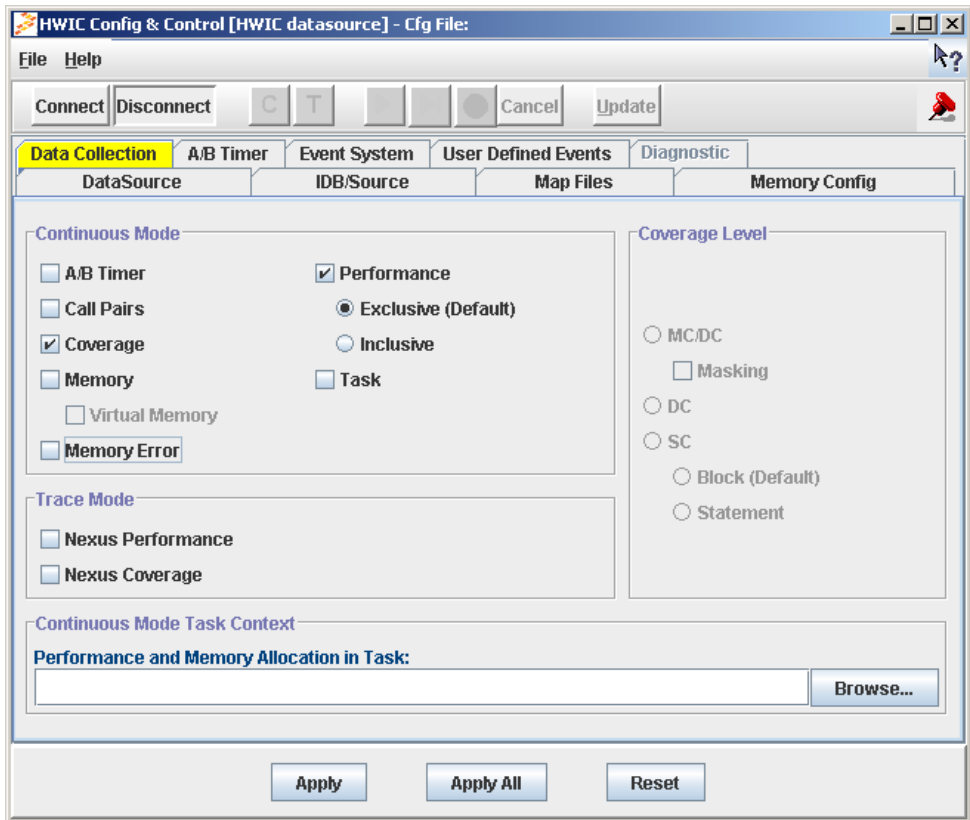
NOTE You cannot select a coverage level unless you have specified an IDB file and applied that setting. How you instrumented your code is reflected in the IDB file, which determines the coverage level selections you can make on the **Data Collection** tab.

4. Click **Apply All**.

For more information on setting coverage levels, see the online Help or “Instrumentation Levels” in the *CodeTEST Instrumenter Reference Manual*.

NOTE If you do not select a coverage level, the default will be the coverage level present in the IDB. If the IDB has files which have been instrumented at different levels, the default is the lowest coverage level present in the IDB. The instrumentation level applied to a source file determines the level of coverage that can be displayed on the screen and in the coverage report. When files or functions have been instrumented at different levels, those at the selected level will display; the others will show “NA.” Change the Manager’s coverage configuration to view files at other levels. The coverage level must also be consistent with available licenses.

Figure 2.5 Data Collection Tab



To save your workspace




1. In the Manager window, select **File > Save Workspace File As...**
2. Browse to the `%CT_HOME%\cttarget\dcu\demo\ct_demo` directory, and type `ct_demo.ctwsp` to name the workspace. Click **Save**.

This name will appear in the title bar of the CodeTEST Manager window. The workspace and configuration data for any defined data sources will be saved.

After you have configured a data source, it is a good idea to save your workspace configuration so you can use it later.

NOTE Saving the configuration in the **Config & Control** window saves only the settings that have been applied from that window.

To start data collection

1. Click the CodeTEST Manager **Connect** button to connect to the Probe.
2. Click the CodeTEST Manager **C**  button to set Continuous mode.
3. Click the CodeTEST Manager **Start**  button to start data collection.
4. Click the CodeWarrior IDE **Run**  button to download and start the CT_DEMO application.

TIP Start CodeTEST data collection before you start your application to get information about how your application starts.

NOTE You must have a valid CodeTEST license for the type of data you want to collect.

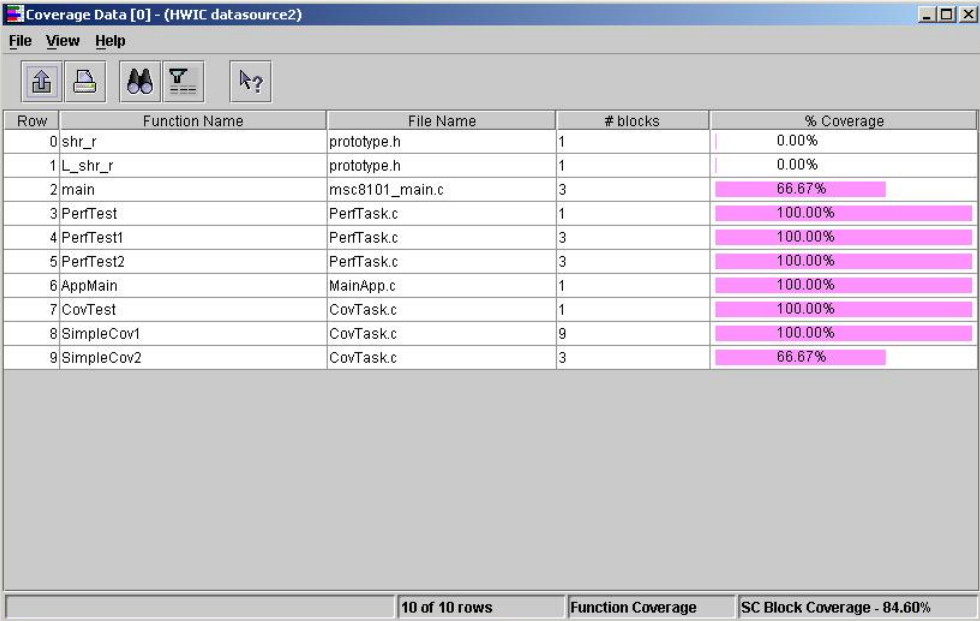
Viewing Data

You can view the results of data collection by double-clicking on the data sets that appear in the Manager workspace pane. Note that when you select one, information about it is shown in the Info pane.

Viewing coverage results

Double-click the **Coverage** data set in the Manager workspace to open the **Coverage Data** view. Nothing displays until the Manager has updated at least once. This may take up to 30 seconds. The results should look similar to the following:

Figure 2.6 Function Coverage View



Row	Function Name	File Name	# blocks	% Coverage
0	shr_r	prototype.h	1	0.00%
1	L_shr_r	prototype.h	1	0.00%
2	main	msc8101_main.c	3	66.67%
3	PerfTest	PerfTask.c	1	100.00%
4	PerfTest1	PerfTask.c	3	100.00%
5	PerfTest2	PerfTask.c	3	100.00%
6	AppMain	MainApp.c	1	100.00%
7	CovTest	CovTask.c	1	100.00%
8	SimpleCov1	CovTask.c	9	100.00%
9	SimpleCov2	CovTask.c	3	66.67%

10 of 10 rows Function Coverage SC Block Coverage - 84.60%

In the **Function Coverage** view, you can view function coverage as well as file coverage. You can sort data by clicking on column headings and you can filter the display from the View menu, or from a single toolbar button. From the **Coverage** view, you can export the data as text, as HTML, or XML.

For more information, see the section “Verifying Code Coverage” in the *CodeTEST Tools User’s Guide*.

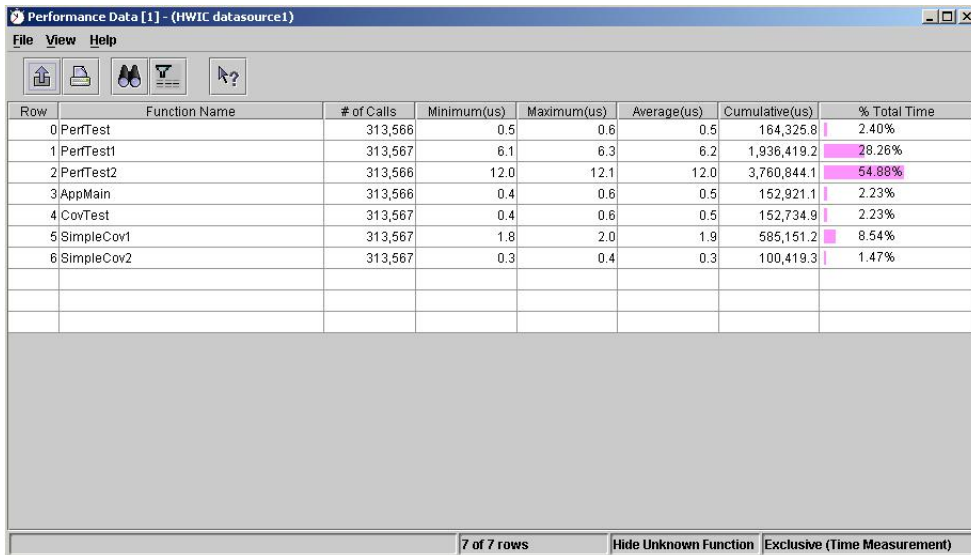
Setting up the Tools

Viewing Data

Viewing performance results

Double-click the **Performance** data set in the Manager workspace to open the **Performance** view. The results should look similar to the following.

Figure 2.7 Function Performance View



Row	Function Name	# of Calls	Minimum(us)	Maximum(us)	Average(us)	Cumulative(us)	% Total Time
0	PerfTest	313,566	0.5	0.6	0.5	164,325.8	2.40%
1	PerfTest1	313,567	6.1	6.3	6.2	1,936,419.2	28.26%
2	PerfTest2	313,566	12.0	12.1	12.0	3,760,844.1	54.88%
3	AppMain	313,566	0.4	0.6	0.5	152,921.1	2.23%
4	CovTest	313,567	0.4	0.6	0.5	152,734.9	2.23%
5	SimpleCov1	313,567	1.8	2.0	1.9	585,151.2	8.54%
6	SimpleCov2	313,567	0.3	0.4	0.3	100,419.3	1.47%

7 of 7 rows Hide Unknown Function Exclusive (Time Measurement)

The **Function Performance** view presents timing and counts information for each function that executes during the measurement, enabling you to compare the relative efficiencies of various portions of your target program.

For more information, see the section “Performance Measurement” in the *CodeTEST Tools User’s Guide*.

Viewing trace results




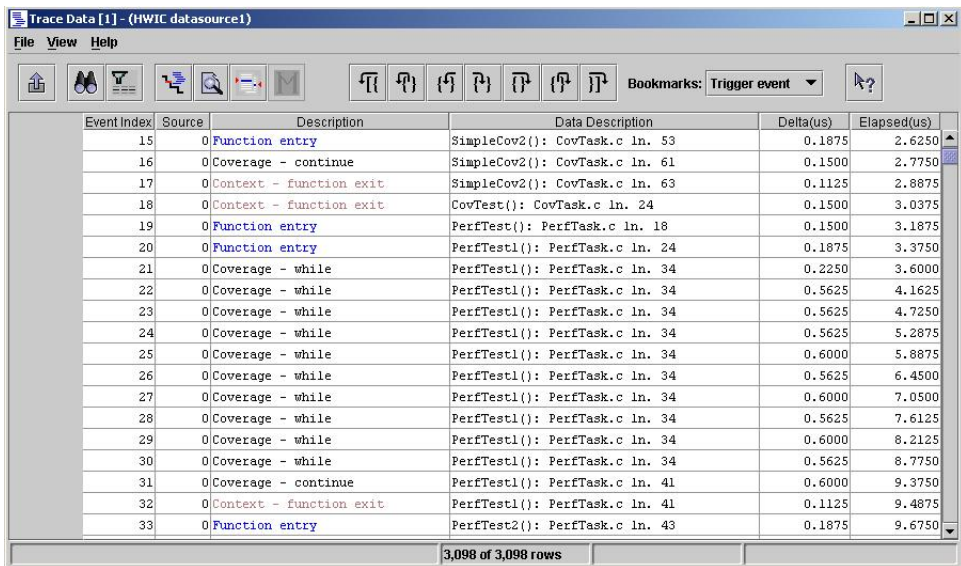
1. From the **Config & Control** toolbar, click  to stop the Probe.
2. Click  to set Trace mode and  to start the Probe data collection.
After a few seconds, the **Trace** data set should appear in the workspace. Data collection stops when the trace buffer fills.
3. Double-click the **Trace** data set to open the **Trace Data** view. The results should look similar to the following.


Figure 2.8 Trace Data View



Event Index	Source	Description	Data Description	Delta(us)	Elapsed(us)
15	0	Function entry	SimpleCov2(): CovTask.c ln. 53	0.1875	2.6250
16	0	Coverage - continue	SimpleCov2(): CovTask.c ln. 61	0.1500	2.7750
17	0	Context - function exit	SimpleCov2(): CovTask.c ln. 63	0.1125	2.8875
18	0	Context - function exit	CovTest(): CovTask.c ln. 24	0.1500	3.0375
19	0	Function entry	PerfTest(): PerfTask.c ln. 18	0.1500	3.1875
20	0	Function entry	PerfTest1(): PerfTask.c ln. 24	0.1875	3.3750
21	0	Coverage - while	PerfTest1(): PerfTask.c ln. 34	0.2250	3.6000
22	0	Coverage - while	PerfTest1(): PerfTask.c ln. 34	0.5625	4.1625
23	0	Coverage - while	PerfTest1(): PerfTask.c ln. 34	0.5625	4.7250
24	0	Coverage - while	PerfTest1(): PerfTask.c ln. 34	0.5625	5.2875
25	0	Coverage - while	PerfTest1(): PerfTask.c ln. 34	0.6000	5.8875
26	0	Coverage - while	PerfTest1(): PerfTask.c ln. 34	0.5625	6.4500
27	0	Coverage - while	PerfTest1(): PerfTask.c ln. 34	0.6000	7.0500
28	0	Coverage - while	PerfTest1(): PerfTask.c ln. 34	0.5625	7.6125
29	0	Coverage - while	PerfTest1(): PerfTask.c ln. 34	0.6000	8.2125
30	0	Coverage - while	PerfTest1(): PerfTask.c ln. 34	0.5625	8.7750
31	0	Coverage - continue	PerfTest1(): PerfTask.c ln. 41	0.6000	9.3750
32	0	Context - function exit	PerfTest1(): PerfTask.c ln. 41	0.1125	9.4875
33	0	Function entry	PerfTest2(): PerfTask.c ln. 43	0.1875	9.6750

3,098 of 3,098 rows

The **Trace Data** view shows the application's flow of execution.

Select the **Source** button to show each statement executed. You can view a graphical version of the trace data that make it easier to identify the relationships between task context and function calls by selecting the **Execution History**  button. All the

trace views are synchronized. When you click on a line in the **Trace Data** or **Execution History** view, the focus changes in the other trace views. You can also synchronize the data between different trace views.

Setting up the Tools

At this Point...

For more information, see the section “Execution Trace” in the *CodeTEST Tools User’s Guide*.

To configure trace acquisition

1. Open the **Config & Control** window, and click the **Event System** tab.
2. Experiment with the following:
 - Set Trigger position. It determines whether data is gathered before or after the Trigger Event.
 - Set Trigger Event. It is the event (for example, function entry) that controls which data is gathered from the trace buffer.
 - Set Trigger Context. It is the execution context (for example, task) within which the CodeTEST Manager recognizes the trigger event.

At this Point...

Now that you have successfully instrumented the CT_DEMO application, you can use the same basic process to set up your own application for your environment. At this point, you have confirmed that:

- The CodeTEST Tools and support files are installed properly.
- The CodeTEST Probe is connected properly.
- The instrumentation setup for CT_DEMO worked for your environment.
- The CodeTEST Manager can communicate with the application and receive data from the Probe.

What to Do Next

We suggest that you perform the following steps to set up your application for the CodeTEST Tools.

Step 1: Instrument your application

Follow the instructions provided for the CT_DEMO application in [“Setting Up for Instrumentation in the CodeWarrior IDE” on page 19](#) to set up your project for the CodeTEST Instrumenter and specify instrumenter options.

Step 2: Set up the Manager for your application

1. Open the Manager. select **File > Load Workspace**, and load the `ct_demo.ctwsp` workspace.

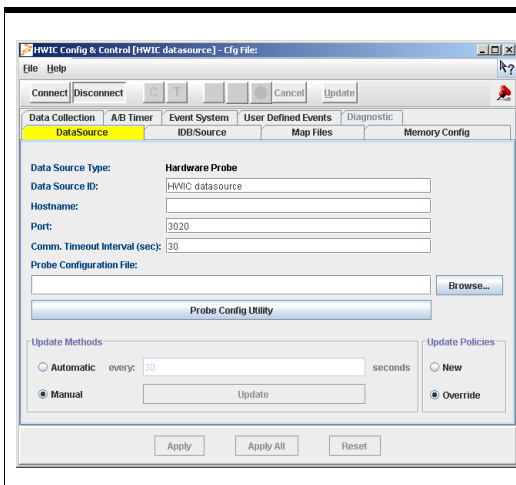
2. In the workspace pane, select and right-click the data source that you created for CT_DEMO. On the context menu, select **Configuration** to open the **Config & Control** window.
3. Click the **IDB/Source** tab and browse to the IDB file that was created when you instrumented your code.
4. Browse to and select each of your source code directories.

This pane allows you to specify any directories that contain source code for the instrumented program, so that you can browse to that code from data windows.

5. Click **Apply All**.

For the CT_DEMO example, we used the default settings for many of the tabs in the **Config & Control** window. When you set up your application, you may want to change these settings. Each tab is described in the following table.

More detailed information is provided in “CodeTEST Data Acquisition” in the *CodeTEST Tools User’s Guide* and in the online Help.

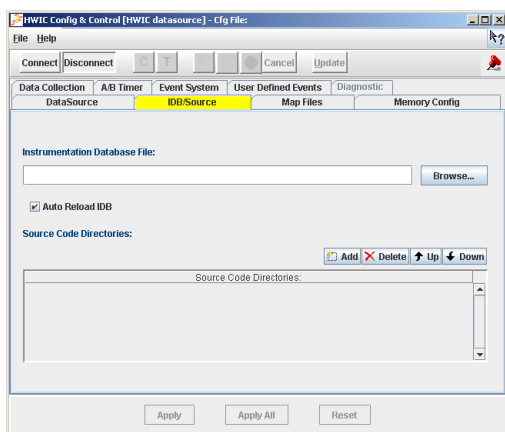


Use the **DataSource** tab to:

- Define and configure the data source.
- Set the Comm Timeout Interval (the interval after which CodeTEST “times out” if it does not receive a response from the Probe).
- Browse to the Probe Configuration File (the *tip* file generated by your Probe in [“Validating the Probe Connection” on page 14](#)).
- Open the Probe Utility to create a Probe tip file.
- Set Update Methods for automatically or manually gathering data from the Probe.
- Set Update Policies to keep or override existing data.

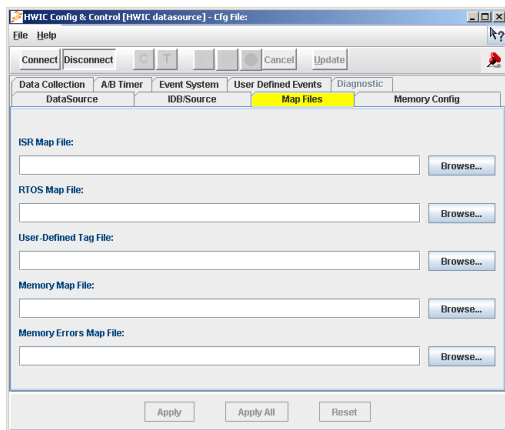
Setting up the Tools

What to Do Next



Use the **IDB/Source** tab to:

- Specify the IDB that was generated when the target program was instrumented.
- Specify the directories that contain your application source code.

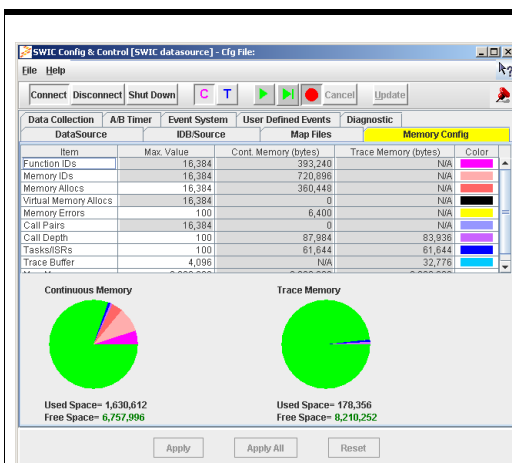


Use the **Map Files** tab to:

- Identify the User Defined Tag file if you have defined your own tags.
- Identify the Memory Map file to display memory. function names (always identify this file if you are monitoring memory).
- Identify the Memory Errors Map file to display specific error names.

Setting up the Tools

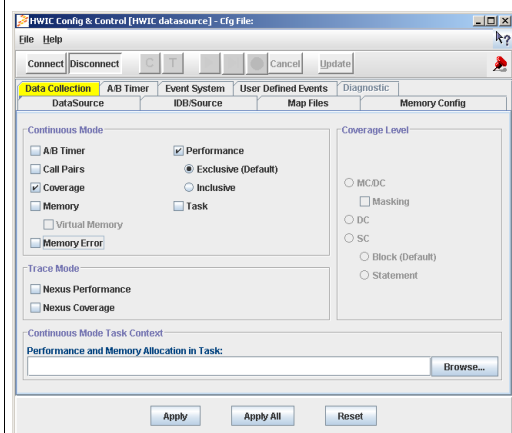
What to Do Next



Use the **Memory Config** tab to:

- Allocate the memory usage of the Data Reduction Processor data bases that are used to process continuous data.
- Configure the maximum size of the trace buffer.

Note: You may need to configure these settings if your application has items that exceed the default CodeTEST limits. (In most cases, the default values will work for your application.) Use these options after connecting to the data source when you are first setting up CodeTEST Tools to monitor your application.

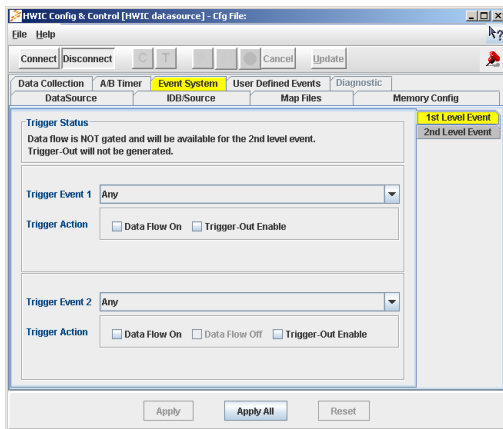


Use the **Data Collection** tab to:

- Select the data types that you want to view.
- Select Coverage levels, including SC (Statement Coverage), DC (Decision Coverage), and MC/DC (Modified Condition Decision Coverage).
- Set Continuous mode setup. This specifies a task to trigger data acquisition of performance data.

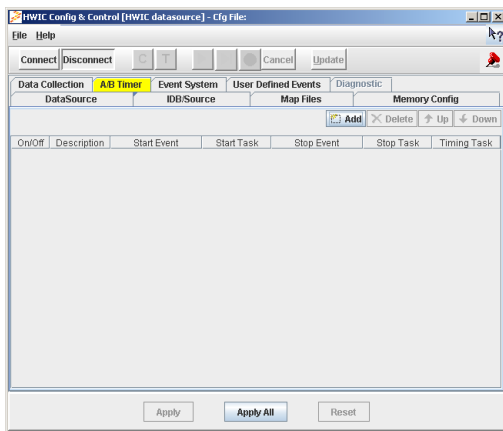
Setting up the Tools

What to Do Next



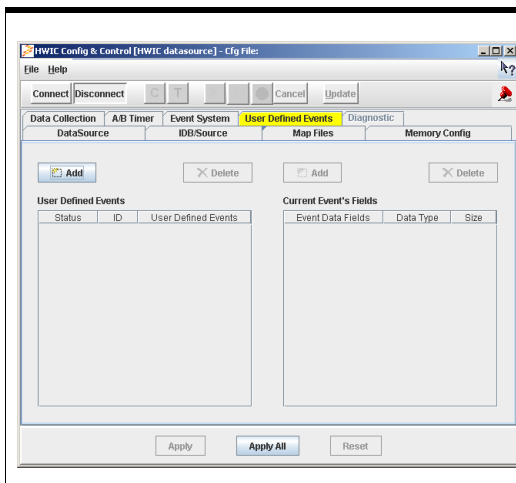
Use the **Event System** tab to:

- Set Trigger Events. This is an event (for example, function entry) that controls which data is gathered from the trace buffer or drives an external trigger signal.
- Set Trigger Context. This is the execution context (for example, task) within which CodeTEST recognizes the trigger event.
- Set Storage Context. This is the execution context (task) within which the data source stores events in the trace buffer.
- Set Trigger position. This determines whether data is gathered before or after the Trigger Event.



Use the **A/B Timer** tab to:

- Define timers to measure the time between two events:
Set a start event and a stop event.
Set the events in the context of a specific task or for any task.
Set timers to measure absolute time or relative (includes only the execution time that is within the context of a specific task) time.
- Turn timers off or on before a measurement.



Use the **User Defined Events** tab to:

- Define events corresponding to function calls you add to your source code. Events can be used in Event System for triggering trace collection and for timing.

Step 3: Start data collection and run your application

1. Connect to the Probe.
2. Load your application onto your target.
3. Start data collection.
4. Start your application.
5. After the Manager has updated the display at least once, open the Coverage, Performance, or Trace data sets to view the results and verify that your application is set up for using CodeTEST Tools.

Step 4: Customize the instrumenting process for your environment

After you have successfully instrumented your code, you may want to use some of the available instrumenting options.

For information about how to...	Go to the following chapters in the <i>CodeTEST Instrumenter Reference Manual</i> ...
Customize the files that configure the instrumenter.	"Configuring Compiler Drivers"
Instrument for SC, DC, and MC/DC Coverage.	"Instrumentation Levels"
Instrument for Performance.	"Instrumentation Levels"
Selectively exclude functions, classes, namespaces, or source files from instrumentation.	"Selective Instrumentation"

Setting up the Tools

What to Do Next

For information about how to...	Go to the following chapters in the <i>CodeTEST Instrumenter Reference Manual...</i>
Use CTPrintf and CTPuts to insert statements similar to printf into your code or create your own user-defined tags. Use CtUserDef to create events that can be used in the Event System and A/B Timers.	"Instrumenting Specific Points of Interest"
Instrument when there are multiple executables sharing common code.	"Instrumenting Multiple Modules"
Instrument in an environment where the components of large software builds are compiled on separate machines or at different times.	"Parallel Builds"
Filter source code before and after instrumentation.	"The CodeTEST Instrumenters"
Use the CodeWarrior IDE to specify instrumentation options.	"CodeTEST Plugin for CodeWarrior IDE Reference" on page 39 of this manual.

For More Information

For information about using CodeTEST Trace, Performance, and Coverage tools, see the *CodeTEST Tools User's Guide*. For more information about using the CodeTEST Manager interface, see the online Help.

CodeTEST Plugin for CodeWarrior IDE Reference

This appendix provides information on using the CodeWarrior Development Tools to set up CodeTEST instrumentation.

- [“Before You Begin” on page 39](#)
- [“Configuring a Project for Instrumentation” on page 40](#)
- [“Troubleshooting” on page 44](#)

You can use the CodeTEST Instrumenter command (`ctci` or `ctcixx`) within a makefile to instrument your code or use the CodeTEST compiler drivers (`ctcc` and `ctcxx`) to control the entire build process and instrument your code. These tools are documented in the *CodeTEST Instrumenter Reference Manual*.

The instrumentation process is simplified when you use the CodeTEST Tools with the CodeWarrior IDE. You can use the **Target Settings Panel** for CodeTEST Instrumentation (see [“CodeTEST Instrumentation Settings Panel” on page 42](#)) to specify instrumentation options. The plugin compiler controls the preprocessing and compiling steps and sends the instrumentation options to the CodeTEST Instrumenter.

Before You Begin

- Install CodeWarrior Development Studio for StarCore v2.6.
- Install the CodeTEST Tools (see [“Before You Begin” on page 11](#)) and set up the environment including the `CT_TARGET` environment variables and the path to the CodeWarrior command-line tools (see [“Preparing Your Environment” on page 13](#)).
- Install any patches, files, or updates provided.
- Configure your project so that it does not use precompiled headers; they are not supported by the CodeTEST Instrumenter.


Configuring a Project for Instrumentation

Targets

To simplify the process of building both uninstrumented and instrumented versions of your project, you may want to create both a standard, uninstrumented target and a target with CodeTEST instrumentation settings.

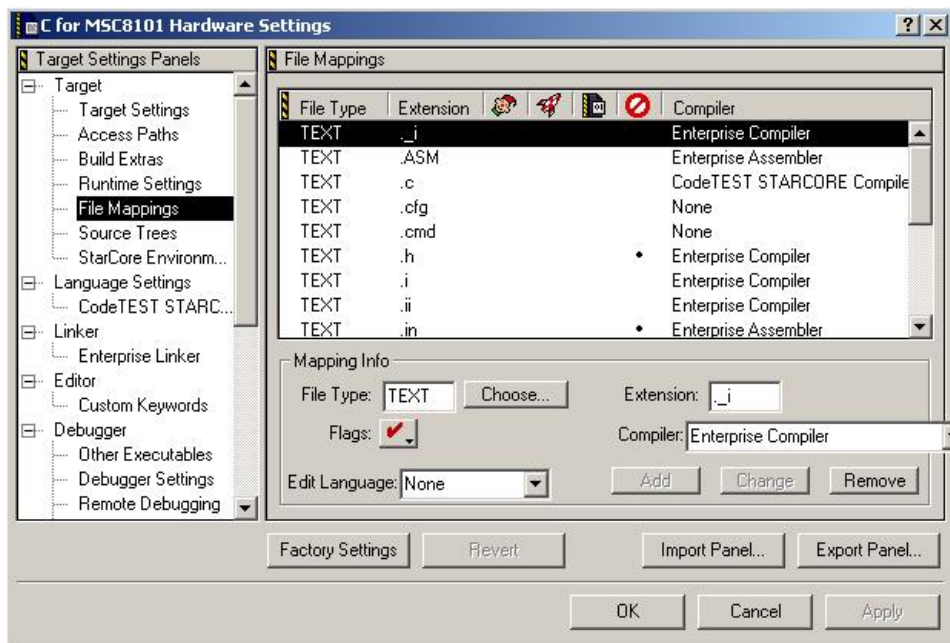
CodeTEST File Mappings Panel

Associate your source file settings with the CodeTEST instrumentation process.

1. Select the CodeTEST target and click the project settings icon .
2. Select **Target > File Mappings**.
3. For each source file type (e.g., .c) that you want to instrument, select **CodeTEST STARCORE Compiler** and click **Change**.
4. Define a new file type:
 - File Type: Text
 - Extension: .i
 - Compiler: Enterprise CompilerClick **Add**.
5. Click **Apply** to apply the settings.


CodeTEST Plugin for CodeWarrior IDE Reference

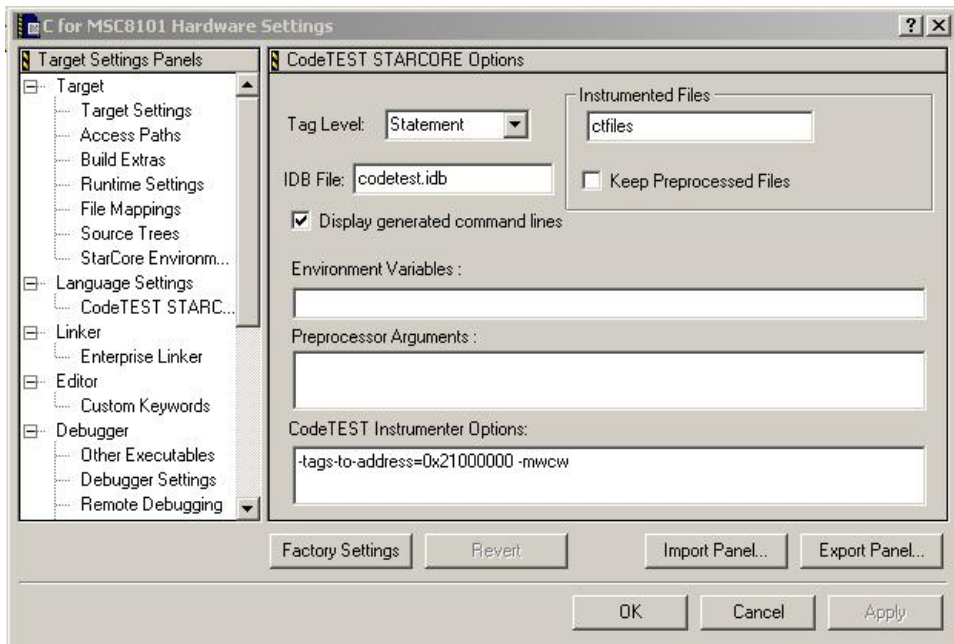
Configuring a Project for Instrumentation



CodeTEST Instrumentation Settings Panel

Specify the instrumentation options for your project in the **Target Settings Panel** for CodeTEST instrumentation.

1. After you open your project, click the project settings icon  and select **Language Settings > CodeTEST STARCORE Options**.
2. Use this panel to select instrumentation options for your project.



The following options are provided in the **CodeTEST STARCORE Options** panel.

See the *CodeTEST Instrumenter Reference Manual* for more information on each option. The text in parentheses after the option name is the Instrumenter command-line equivalent.

Tag Level (-tag-level=*level*)

This option controls the number of instrumentation tags inserted into an application and the type of data gathered. (*See also*: “Instrumentation Levels” in the *CodeTEST Instrumenter Reference Manual*.)

- **Disabled**: disables instrumentation.
- **Performance**: instruments for Performance and Trace data.

- **Statement:** instruments for Performance, Trace, and Statement Coverage data.
- **Decision:** instruments for Decision Coverage data.
- **Extended Decision:** instruments for extended Decision Coverage data.
- **MC/DC:** instruments for Decision Coverage and Modified Condition/Decision Coverage data.
- **Extended MC/DC:** instruments for Decision Coverage, and extended Modified Condition/Decision Coverage data.

Instrumented Files

This option preserves instrumented files in the named directory. The named directory is located in the same directory as the source files.

Keep Preprocessed Files

This option preserves temporary preprocessed files in the directory named in the **Instrumented Files** field.

IDB File (**idb=file**)

This option specifies the instrumenter database (IDB) name. If left blank, the file is named `codetest.idb` and written to the target output directory. You can enter either a filename or a path and a filename. If you change the filename after it is created, a new IDB file is created the next time you compile. The old file is retained in case you need to use it again.

Environment Variables

This option lets you define CodeTEST environment variables, `CT_HOME` and `CT_TARGET`, for this target build. Syntax: `ENV_VAR=path`.

Preprocessor Arguments

This option lets you specify command-line arguments for the preprocessor for files mapped to the CodeTEST Instrumenter.

CodeTEST Instrumenter Options

This option lets you specify additional command-line arguments for the CodeTEST Instrumenter. They are passed at the end of the command line, before the final source file and destination file arguments, to allow overrides of options implied by the Target Settings Panel. The string must not be more than 256 characters. You can specify a command file, using response file syntax: `@arguments.txt`. You must fully specify the file, either with a full path or a source tree. For extensive information about

CodeTEST Plugin for CodeWarrior IDE Reference

Troubleshooting

instrumentation options, see the *CodeTEST Instrumenter Reference Manual*. See also the *Instrumenter Command Quick Reference Guide*.

Use the option `-mwcw` to specify the StarCore Enterprise compiler to the CodeTEST Instrumenter.

Troubleshooting

Compiler errors

Failure to find a current CodeWarrior license can result in compiler error warnings. Verify that the `license.dat` file exists and can be found by the license manager.

Instrumenter Not Found Message

Verify that your environment is set up correctly. See [“Preparing Your Environment” on page 13](#).

Warning during DC and MC/DC instrumentation

When instrumenting for DC and MC/DC, the compiler may generate the following warning:

```
Warning:C5909: Assignment in Condition.
```

You can safely ignore this warning can if it appears only when you compile instrumented code, but not when you compile the same uninstrumented code.

Syntax errors on long long types

Should the compiler generate a syntax error regarding the `long long` type, such as:

```
Error : declaration syntax error
main._i line 342 long long quot;
```

you can use this workaround. In the **CodeTEST STARCORE Options** panel, add the following in the **Preprocessor Arguments** field:


```
-D_MSL_LONGLONG=0
```

Instrumenting for Memory Analysis

This appendix provides instructions on instrumenting for memory analysis.

You will need to build a CodeTEST support library and add it to your project. You will also need to define the CodeTEST tag ports.

To build the support files

1. Start the CodeWarrior IDE for StarCore.
2. Select **File > New**.
3. In the **New** dialog box, select **StarCore Stationery**.
4. Enter CTMem (or another name of your choice) for **Project name** and click **OK**.
5. Select the options appropriate for your target hardware and click **OK**.
6. When the project opens, select the target appropriate for your system.
7. Click the project settings icon  .
8. Select **Linker > Enterprise Linker**.
9. Change the **Output File Name** to `ctMem.elb`.
10. Click **OK**.
11. Copy `%CT_HOME%\cttarget\common\include\cttagport.h` and `%CT_HOME%\cttarget\common\src\ctwrpc.c` into the CTMem project `src` directory.
12. On the project **Files** tab, remove all filenames listed.
13. In the project **Files** tab, under **Sources**, add `cttagport.h` and `ctwrpc.c` to the project.
14. Modify `cttagport.h` as follows, and save the file. After the line `#define TAG_TO_PTR 3` add `#define TAG_DEST 4:`


...

`#define TAG_TO_PTR 3`

Instrumenting for Memory Analysis


```
#define TAG_DEST    4
```

15. Ensure that all other target settings match those for your application.

16. Click the **Make** button  to build the project.

`ctMem.a` is built in the project `bin` directory.

To use the library

1. In the project you are instrumenting select the appropriate target and click the project settings icon .

2. Select **Linker > Enterprise Linker**.

3. In the **Additional Options** edit box, add the following: `-lctmem.elb`

4. Make sure that the `ctmem.elb` library is in your Library Path.

5. Select **Language Settings > CodeTEST STARCORE Options**.

6. In the **CodeTEST Instrumenter Options** edit box, add the following to the existing entries:

```
-tag-allocator=%CT_HOME%\cttarget\common\map\ctwrap.map
```

Substitute the path to the CodeTEST Tools installation for `CT_HOME`.

To define the tag port

In your linker command file, add the following:

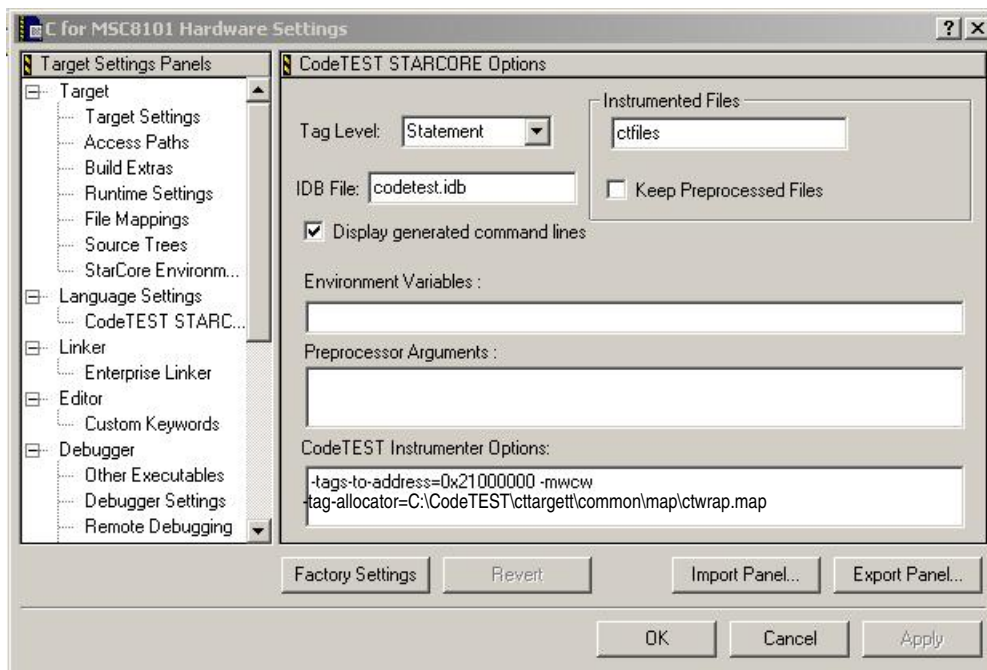
```
.provide _ct_port, 0x21000000
```

Change the address specified to one appropriate for your target hardware.

See “The CodeTEST Tag Format” in the *CodeTEST Instrumenter Reference Manual* for more information on tag port options.

NOTE If you are using the `ct_demo` application, it is still instrumented using the `-tags-to-address` mode for the tag port format, but the `ctMem` library uses the `ct_port` variable to write the tags. Therefore, `ct_port` needs to be defined. The address of `ct_port` must be the same as the one used for `-tags-to-address` instrumentation. If you choose to not use `-tags-to-address` to instrument `ct_demo` or your application, but you choose to use `tags-to-port` default instrumentation, then the `ct_port` definition in the lcf file will be used by both the application and the `ctMem` library.

Figure B.1 CodeTEST STARCORE Options Panel



To build the project

1. Ensure that the file mappings are set appropriately for CodeTEST instrumentation, as described in [“To map source file extensions for CodeTEST instrumentation” on page 21](#).

2. Click the **Make** button  to build the project.

Now when you collect data using the CodeTEST Probe, a memory data set will be created.

To configure the CodeTEST Manager

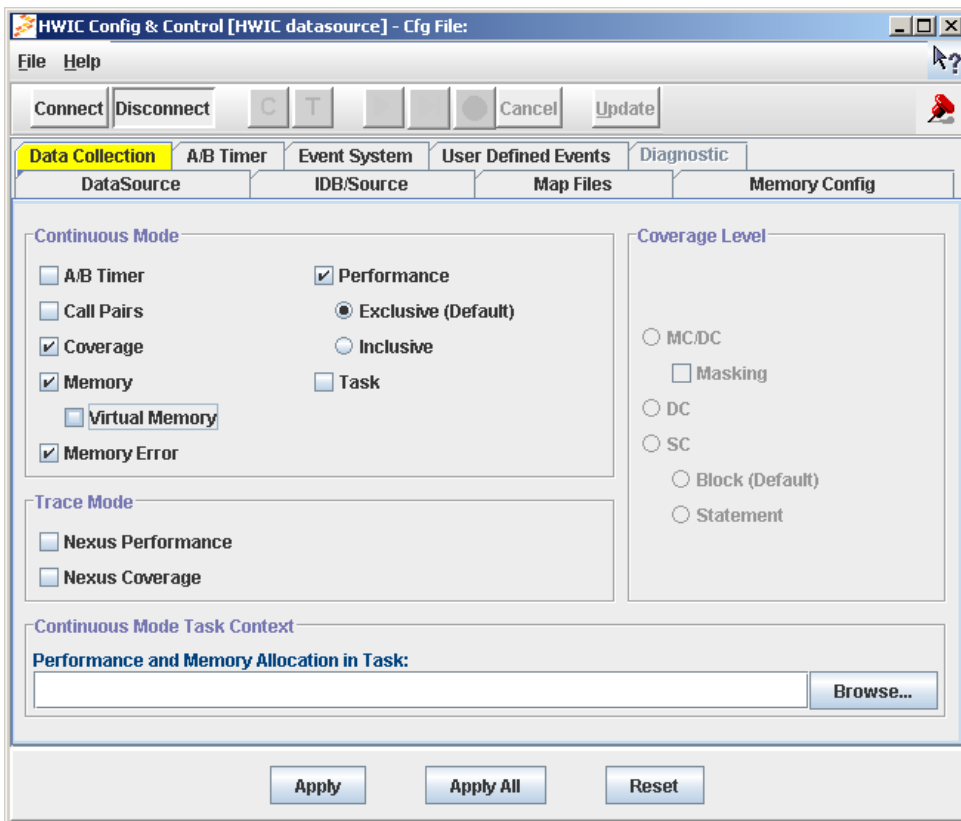
When configuring your CodeTEST data source for memory analysis, perform the following steps.

1. In the **Config & Control** window for your data source, click the **Data Collection** tab.
2. Select **Memory** and **Memory Errors**.
3. Click **Apply**.
4. Click the **Map Files** tab.
5. In the **Memory Map File** field, browse to the default memory map file, `%CT_HOME%\cttarget\common\map\ctwrap.map`, and select it.

This enables display of the names of the original memory functions (`malloc`, etc.), as well as the function names that called the memory allocation/deallocation. If a map file is not specified, the **Type** column of the **Memory Data** view displays hex numbers instead of memory function names.

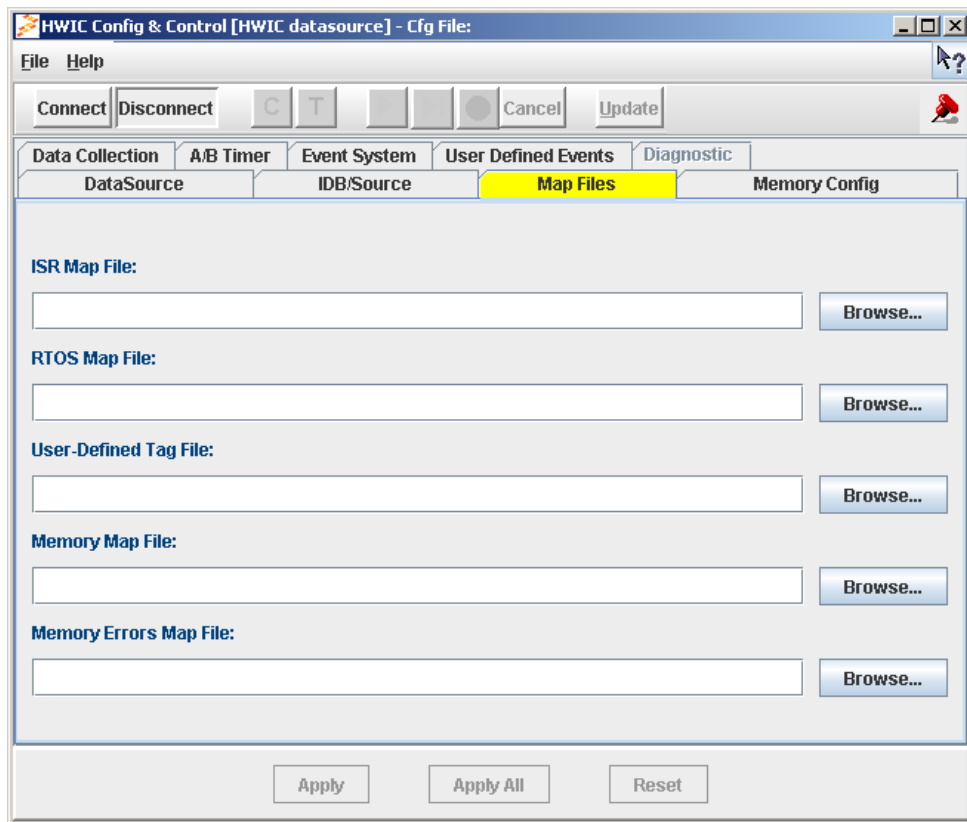
6. In the **Memory Errors Map File** field, browse to the memory errors map file, `%CT_HOME%\cttarget\common\map\cterr.map`, and select it.
7. Click **Apply**.

Figure B.2 Data Collection Tab



Instrumenting for Memory Analysis

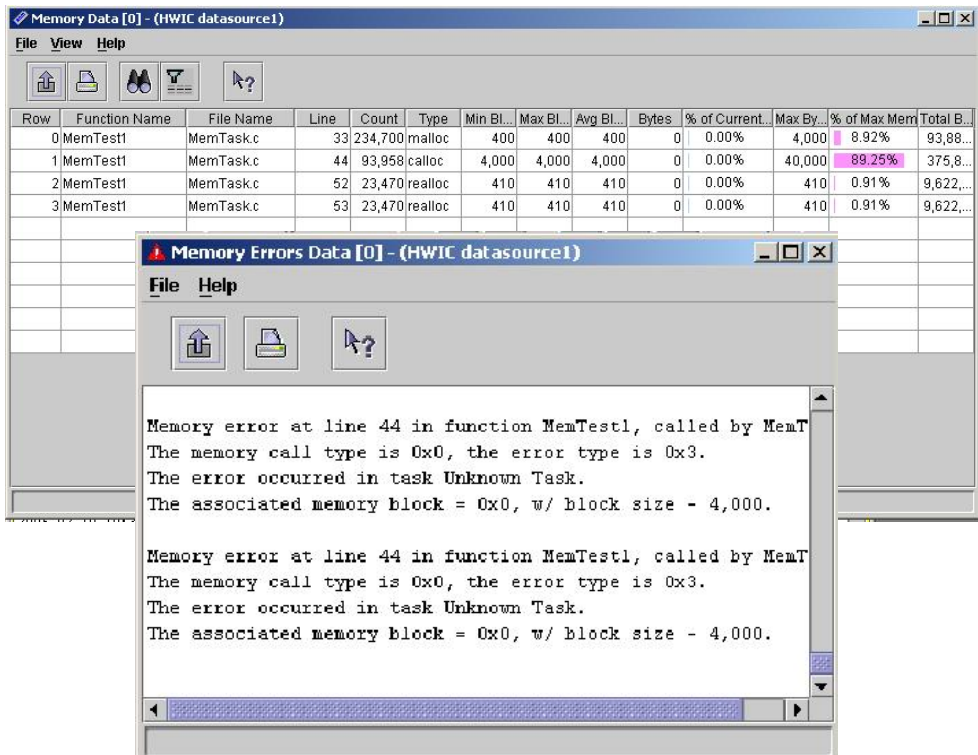
Figure B.3 Map Files Tab



To view memory results

After collecting data, in the CodeTEST Manager, double-click the **Memory** data set in the Manager workspace to open the **Memory Data** view. See the following sample **Memory Data** view and **Memory Errors Data** view.

Figure B.4 Memory Data Views



The **Memory Data** view tracks your application's dynamic allocation and deallocation of memory. Each row in the **Memory Allocation** table represents an allocation caller (that is, a specific location in your target code where a memory allocation routine is called).

This view provides a number of options for presenting the data. From the **View** menu select **Columns** and check desired columns; from the **View** menu select **Function Name** and then click **Short name** or **Long name**.

For more information, see "Measuring Memory Usage" in the *CodeTEST Tools User's Guide*.



Instrumenting for Memory Analysis

Index

A

A/B Timer tab 36
A/B Timers view 6

C

Call Pair view 6
Cancel button 17
CodeTEST
 driver messages 24
 features 6
 setting up 9
 toolbar 16
 Tools 6
 using 9
CodeTEST compiler driver 39
CodeTEST Instrumenter 39
CodeTEST Instrumenter Options 43
CodeTEST Manager 15, 24
CodeTEST StarCore Compiler 21
codetest.zip 13
CodeWarrior
 instrumenting 19, 40
 plugins 13
 project 19, 40
 Tools 11, 39
 using with CodeTEST 11, 39
 version 39
CodeWarrior Development Studio for
 StarCore 12
CodeWarrior IDE 10
CodeWarrior StarCore compiler 11
command-line arguments
 Instrumenter 43
compiler 11
compiler driver 39
Config & Control window 16
configuration
 system 8
connect
 to Probe 28
Connect button 16

connection method 14
Continue button 17
Continuous mode 28
Continuous mode button 16
coverage 29
 Decision (DC) 6
 levels 43
 Modified Condition/Decision Coverage
 (MC/DC) 6
 specifying level 26, 42
 Statement(SC) 6
Coverage Level 26
coverage tool 6
Create HWIC Data Source 17
CT_DEMO 10, 11, 12
 instrumenting 19
CT_HOME 12
ct_port 15, 21, 23, 47
CT_TARGET 14
ctcc 39
ctci 39
ctcixx 39
ctcxx 39
cterr.map 48
ctMem library 47
cttagport.h 45
ctwrap.map 46
ctwrpc.c 45
Customer Support 2

D

data
 viewing 29
data collection 9
Data Collection tab 26, 35, 48
data sets 15
data source 15
data types 26
database
 instrumenter 43
DataSource tab 33
debug mode 24

- Decision Coverage 43
- decision instrumentation level 43
- demo application 12
- development environment 11
- diagnostics 19
- Disabled instrumentation level 42
- Disconnect button 16

E

- environment
 - setup 13, 14
- environment variable 14, 43
- Event System tab 36
- Execution History 31
- execution history 6
- Execution History button 31
- Execution History view 6
- Extended Decision Coverage 43
- Extended Modified Condition/Decision Coverage 43

F

- features 6
- File Mappings Panel 21, 40
- file types 21, 40
- Freescall Support 2
- Function Coverage view 29
- Function Performance view 30

H

- Hardware Probe 7
- Hostname 17

I

- IDB
 - coverage 27
 - specifying in Manager 25
- IDB file 43
- IDB/Source tab 25, 34
- idb=file 43
- Info pane 15
- installation
 - requirements 11

- instrumentation levels 42
- Instrumentation Settings Panel 22, 42
- Instrumenter 9, 39
 - command-line arguments 43
- instrumenter database 43
- Instrumenter options 22, 23, 42
- Instrumenter plugin files 13
- IP addresses 12

K

- Keep Preprocessed Files 43

L

- lcf 46
- Library Path 46
- linker command file 46
- Logs pane 15

M

- Manager 15, 24
- manuals 5
- Map Files tab 34, 48
- mapping files 21
- MC/DC instrumentation level 43
- measurements
 - coverage 6
 - memory 6
 - performance 6
 - trace 6
- memory
 - displaying results 51
- memory analysis
 - instrumenting for 45
- Memory Config tab 35
- Memory Errors Map File 48
- Memory Errors view 6
- Memory Map File 48
- memory support files 45
- memory tool 6
- memory usage 35
- Memory view 6, 51
- Modified Condition/Decision Coverage 43

N

no instrumentation 42

O

online help 6

overview 5

P

patches 39

path 14

performance 6, 29

performance data 30

Performance instrumentation level 42

Performance view 6

plugins 13

port address 15, 21, 47

port options 46

precompiled headers 39

preprocessor arguments 43

prerequisites 11

Probe 7

Probe configuration 18

Probe connection 19

Hardware Probe 14

Probe Utility 18

procedure

overview 13

product introduction 5

project

configuring for instrumentation 19, 40

instrumentation options 22, 42

requirements 39

S

saving a workspace 28

setup

environment 13

requirements 11

steps 13

software manuals 5

Source Code view 6

source file types 21, 40

source files

specifying 26

StarCore compiler 11

StarCore family support 5

StarCore stationery 12

starcore.zip 13

starcore-CodeWarrior.zip 13

Start button 17

Statement Coverage + Performance 43

stationery 12

Stop button 17

support files

memory 45

system

diagram 8

T

tag port 15, 21, 23, 47

tag port options 46

-tag-level=level 42

Target Settings Panel 39

taskwalk 14

Technical Support 2

timers 36

toolbar buttons

Cancel 17

Connect 16

Continue 17

Continuous 16

Disconnect 16

Start 17

Stop 17

Trace 16

Update 17

tools 6

trace data 29

collecting 32

Trace Data view 31

Trace mode 31

Trace mode button 16

trace tool 6

Trace view 6

troubleshooting

instrumenter not found 44

U

- Update button 17
- Update Methods 17
- updates 39
- User Defined Events tab 37
- using the Manager 24

V

- validating the Probe connection 14

W

- web address 2
- workspace
 - saving 28
- Workspace pane 15