

*Dreamcast  
CodeScape  
for Set 5  
User Guide*

---

## Legal Notice

### IMPORTANT

The information contained in this publication is subject to change without notice. This publication is supplied "as is" without warranty of any kind, either express or implied, including but not limited to the implied warranties or conditions of merchantability or fitness for a particular purpose. In no event shall Cross Products be liable for errors contained herein or for incidental or consequential damages, including lost profits, in connection with the performance or use of this material whether based on warranty, contract, or other legal theory.

This publication contains proprietary information which is protected by copyright. No part of this publication may be reproduced in any form, or stored in a database or retrieval system, or transmitted or distributed in any form by any means, electronic, mechanical photocopying, recording, or otherwise, without the prior permission of Cross Products Limited.

## CodeScape User Guide

### Revision History

Version 2.2.0 build 121, March 1999

Release Candidate 1, 7 October 1996, beta 2, 26 August 1996, beta 1, 26 July 1996 - 97, 2.0.0, March 1998, Version 2.0.5a, May 1998, Version 2.1.0. alpha build 86, July 1998, Version 2.1.2. beta build 94, October 1998

© 1999 Cross Products Limited. All rights reserved.

Microsoft, MS-DOS, and Windows are registered trademarks, and Windows NT and JScript are trademarks of Microsoft Corporation in the United States and other countries. CodeScape and SNASM are registered trademarks of Cross Products Limited in the United Kingdom and other countries. Brief is a registered trademark of Borland International. CodeWright is a registered trademark of Premia Corporation. Multi-Edit is a trademark of American Cybernetics, Inc. All other trademarks or registered trademarks are the property of their respective owners.

---

# Contents

<b>Before you begin</b>	<b>5</b>
Document conventions	6
This guide	7
The CodeScape software	8
<b>Using and configuring the interface</b>	<b>9</b>
The commands on the menu bar	10
Customizing shortcut keys	12
The commands on the toolbars	14
Commands on each toolbar	17
<b>How windows and regions work</b>	<b>27</b>
Using windows	28
Using regions	30
Configuring regions	34
Target window	36
Target Processor display	37
Input/Output window	38
The Source and Disassembly regions	41
The Call Stack region	51
The Watch and Local Watch regions	52
The Watch region	54
The Local Watch region	58
The Memory region	61
The Register region	69
Hitachi target processor register region display	74
The Edit region	77
Opening and saving files	79
Search and replace	80
Cutting and pasting text	82
Using bookmarks	83
<b>Interacting with target processors</b>	<b>85</b>
Connecting to a target processor	86
Add files to a project	88
Restarting a program	91
<b>Working with sessions</b>	<b>93</b>

---

## **Contents**

---

<b>Working with projects</b> .....	<b>97</b>
Setting up a project build environment .....	98
Setting up an editor .....	100
Setting up the project commands .....	102
<b>Debugging</b> .....	<b>105</b>
Debugging modes .....	106
Running and stopping programs .....	107
Stepping into (tracing) code .....	110
Breakpoints .....	116
Configuring breakpoints .....	121
Breakpoint expression format .....	129
<b>Simulating a target processor</b> .....	<b>135</b>
Information generated by the Simulator .....	139
Reading the results of simulation .....	144
<b>Profiling program files</b> .....	<b>145</b>
Using the profiler: an overview .....	146
The Profiler's commands .....	148
<b>Viewing GD-M log information</b> .....	<b>155</b>
<b>Writing scripts to automate tasks</b> .....	<b>157</b>
Scripting commands .....	160
<b>Evaluating expressions</b> .....	<b>179</b>
C/C++ expressions .....	182
Assembler expressions .....	185
<b>Using the command-line</b> .....	<b>187</b>
<b>Appendix A: Frequent operations</b> .....	<b>191</b>

---

# ***Before you begin***

---

## **This release**

The CodeScape version 2.2.0 build 121 release includes:

- A CD-ROM that contains the CodeScape debugger, and online documentation including context-sensitive help.
- A hardware setup guide, and a software reference manual.

*NOTE: Contact Technical Support if any of these items are missing.*

## ***Before you begin***

---

### **Document conventions**

*NOTE: Notes call attention to important features or instructions.*

*Typographic conventions in this guide*

Convention	Description
SPACEBAR	Capital letters denote the names of keys on the keyboard, filenames, and extensions.
ALT+F4	If two or more keys must be pressed simultaneously, they are linked with a plus sign (+).
ALT, F, X	If two or more keys must be pressed in succession, the keys are linked with a comma (,).
<i>select this box</i>	Italics denote text boxes and check boxes that are on CodeScape's interface.
emphasis	Bold text denotes emphasis.
input and output	This font denotes user input and program output, including error messages.
<b>command-line</b>	This font denotes command-line options.

### **Audience**

This manual is for programmers that write for targets under Windows® 95, Windows® 98, or Windows NT™ 3.51/4.0. It will also be of use to technical support and software test engineers.

### **This guide**

**Using and configuring the interface** introduces CodeScape's debugging environment and how to use it. It describes the commands on the menu bar, toolbars, and shortcut menus. It explains how to set up and use windows and regions for your project.

**Interacting with target processors** explains how to connect to, initialize and reset a target processor. This includes: restarting a program, and saving and loading binary parts of a program.

**Working with sessions** explains how to use the commands for working with sessions. This includes how to: open new and existing sessions, save a session, save a session with a new name, close a session, view and use the recently used files list, and exit CodeScape.

**Working with projects** describes how to set up and use your project build environment including how to set up an MS-DOS or Windows editor. It also tells you how to build your project within CodeScape and what to do if CodeScape returns errors after a compile.

**Debugging project files** explains the various ways you can debug your project files, including: stepping source code, setting watch and data breakpoints, and simulating assembly code.

**Simulating a target processor** describes how to use the Simulator to obtain data that will help you to optimize timing critical sections of Assembly code.

**Profiling program files** explains how to use the Profiler to examine the run-time behavior of program files written for Hitachi SH series processors.

**Viewing GD-M log information** describes the GD-Workshop log events you can control within CodeScape.

**Writing scripts to automate tasks** explains how to use CodeScape's script commands to automate routine tasks.

**Evaluating expressions** explains how to use the Expression Evaluators and describes all of the C/C++ and Assembler expressions that are supported.

**Using the command-line** describes all of the command-line commands.

## ***Before you begin***

---

### **The CodeScape software**

CodeScape is a fast, intuitive, Windows-based development environment. CodeScape's debugging features let you find, isolate, and fix bugs in your original source, or disassembled code.

Run CodeScape to:

- 1) Edit your project files.
- 2) Compile and link your project files.
- 3) Debug your project and test it for errors.
- 4) Then do one of the following:
  - If debugging returns errors, repeat steps 1 to 3 above.
  - OR-
  - If debugging does not return errors, Build your project.

To run, CodeScape requires:

- An IBM™ PC or compatible with Pentium™ 90 processor or above.
- Windows® 95, Windows® 98, or Windows NT™ 3.51/4.0.
- 32 MB of RAM minimum, 128 MB of RAM recommended for program files with a debug section that is 24 MB or above.



# ***Using and configuring the interface***

---

You can control CodeScape using a mouse or keyboard. CodeScape has many useful toolbars that can be docked, floating, or hidden. Region specific shortcut menus are available for the most used functions.

The user interface consists of:

- The Menu bar.
- Toolbars.
- Windows.
- Regions.

### **The commands on the menu bar**

#### **File menu ALT+F**

The commands in the File menu are for working with sessions, resetting the target, loading program files, restarting program file execution, and saving and loading binary information. Use Save binary and Load binary to move large blocks of data in and out of memory. The File menu also displays a list of recently used session files. You cannot hide this list or change the number of files displayed.

*NOTE: When you load a new session, or exit CodeScape, a message appears prompting you to save any changes to the current session.*

#### **Edit menu ALT+E**

The commands in the Edit menu are for cutting and pasting in the Edit region, and for performing searches. The Edit menu becomes available when you open a window, create a region, or load a session.

#### **View menu ALT+V**

The commands in the View menu are for showing and hiding toolbars, and configuring regions.

#### **Project menu ALT+P**

The commands in the Project menu are for configuring the current project and building it from within CodeScape.

#### **Debug menu ALT+D**

The commands in the Debug menu are for controlling program execution, stepping code, and using breakpoints. You can set the cursor to the PC (program counter) and vice-versa. The default origin is set to the value of the PC. You can also lock the view to an Expression that contains the PC, a register, or memory.

#### **Tools menu ALT+T**

The commands in the Tools menu let you simulate a target's processor operations on your computer and run the Profiler to examine the run-time behavior of your program file.

Other commands are for running script files and adding script files to run from the Tools menu and the shortcut menu on the Scripts tab on the Input / Output window. You can also specify custom shortcut keys, and add programs to run from the Tools menu.

### Region menu ALT+R

The commands in the Region menu are for splitting (creating new) regions, changing a region's type, and updating all regions. The Region menu is available when you open or create a window, region, or session.

### Window menu ALT+W

Use the Window menu to open a new window. When you open a new window, the Edit and Region menus appear on the menu bar, and additional commands appear on the Window menu which are for arranging multiple windows in CodeScape. You can select and deselect commands for proportionally resizing a window and its regions, and loading the current session when you next run CodeScape.

The Window menu also displays a list of region types and highlights the currently active region. When you have more than one region in a window frame, the active region within that frame appears in the list. You cannot hide this list or change the number of regions displayed.

*NOTE: If you use a Windows® 95 or a Windows® 98 machine, creating too many new windows or too many new regions causes CodeScape to run out of system resources.*

### Help menu F1

Use the Help menu to get on-line Help and view CodeScape version information.

### Region specific shortcut menus

Each region has two shortcut menus:

- 1) The Region Type menu has commands for changing the region's type. To see the menu:
  - Press CTRL+SHIFT+F10.
  - OR-
  - CTRL+Right-click anywhere in the region.
- 2) The Region Actions menu has region specific commands, and global commands for controlling program execution or manipulating breakpoints. To see the menu:
  - Press SHIFT+F10.
  - OR-
  - Right-click anywhere in the region.

### **Customizing shortcut keys**

You can specify shortcut keys for any of the commands on the menu bar or on the shortcut menus.

When you assign a shortcut key to a command CodeScape saves the setting in Keyboard File, CODESCAPE.MAC. Each time you change a keyboard shortcut CodeScape updates the information in the Keyboard File. When you run CodeScape it automatically detects and loads the Keyboard File, CODESCAPE.MAC.

You can save your settings to a Keyboard File with a specific name. This is useful if you use CodeScape on more than one computer, or if you share a computer with another person.

To save your settings to a Keyboard File with a specific name:

- 1) On the Shortcut Keys dialog box create and remove any shortcut keys you require, then click Save...
- 2) Enter a name for the Keyboard File using the extension \*.mac, then click OK.

*NOTE: To load a Keyboard File, click Load... and enter the filename and location of the Keyboard File that you want to use. When you load a Keyboard File the settings it specifies are automatically copied to CODESCAPE.MAC.*

You can:

- Assign shortcut keys to a command.
- Remove shortcut keys from a command.
- Restore the shortcut keys to their original settings.
- Assign the Microsoft Developer Studio shortcut keys to the commands.

### Assigning shortcut keys to commands

- 1) Click Tools, select Customize then click Keyboard...  
The Shortcut Keys dialog box appears.
- 2) In the Select a command tree highlight a menu command to assign a shortcut.  
A description of the command appears in the Descriptions box, and any shortcut keys appear in the Assigned shortcuts box.
- 3) Click Create Shortcut...  
The Select shortcut dialog box appears.
- 4) Press the key combination you require.  
If you press a key or key combination that is currently assigned to another command, that command appears under Replaces.
- 5) Click OK.  
The new shortcut appears in the Assigned shortcuts text box.
- 6) Click OK.

### Removing shortcut keys from commands

- 1) Click Tools, select Customize then click Keyboard...  
The Shortcut Keys dialog box appears.
- 2) In the Select a command tree highlight a menu command.  
A description of the command appears in the Descriptions box, and any shortcut keys appear in the Assigned shortcuts box.
- 3) Click Remove.

### Restoring shortcut key assignments to their original settings

- 1) Click Tools, select Customize then click Keyboard...
- 2) Click CodeScape Defaults.

### Assigning Microsoft Developer Studio shortcut keys to commands

- 1) Click Tools, select Customize then click Keyboard...
- 2) Click DevStudio compatible.

*NOTE: For menu items that do not have a Microsoft Developer Studio default value, for example Simulate Processor, the CodeScape default shortcut key is used.*

### The commands on the toolbars

The toolbars provide access to the main debugging functions. To use the *Toolbar Configuration* check box to show or hide toolbars click View, Toolbar, then select or deselect toolbars from the list.

#### *Toolbars and their uses*

Use the:	To:
Breakpoint toolbar	Access the most common breakpoint actions.
Debug toolbar	Access the debugging actions.
Processor Combo toolbar	Select a target processor, configure a target processor, and load program files.
Input/Output window	View: the build utility's output when you build a project; all messages generated by the target; and all messages generated by an executing script.
Region toolbar	Set and change a region's type.
Region Combo toolbar	Set the rate at which a region's display is updated, and change a region's type.
Splitter toolbar	Split regions using the mouse.
Standard toolbar	Open new windows, and create, open, and save sessions.
Target window	View the active processor for the selected target, load program files and configure the target.
Target Combo toolbar	View and configure the active target.
Editor toolbar	Use the editing actions.
Workshop toolbar	Use the GDWorkshop log options.
Status Bar	View contextual information about commands on the interface and any other information about the current state of the interface.

**NOTE:** *The Target and Input/Output windows can be docked at the top and bottom of the main window, or left free floating.*

## View, hide, dock, and move toolbars

*NOTE: For more information see Toolbars and their uses.*

### View toolbars


Do one of the following:

- Right-click the status bar.  
-OR-
- Right-click on a blank area of any toolbar. Select the toolbar.  
-OR-
- Click View, Toolbar... Select the *toolbar* check box. Click OK.

### Hide toolbars

- 1) Right-click on a blank area of any toolbar.
- 2) Clear the toolbar from the list.

If the toolbar is undocked:

- Right-click on the toolbar title bar and click Hide.  
-OR-
- On the toolbar title bar, click  .

If the toolbar is docked:

- 1) Click View, then point to Toolbar...
- 2) Clear the *toolbar* check box. Click OK.

### Dock toolbars

Do one of the following:

- Drag the toolbar to an edge of the main window.  
-OR-
- Double-click the title bar. The toolbar will be docked at its last docked position.

## ***Using and configuring the interface***

---

### **Move toolbars**

- 1) Do one of the following:
  - On the toolbar title bar, right-click and click Move.
  - OR-
  - Click the toolbar title bar.
- 2) Drag the toolbar to the required position.

*NOTE: You cannot dock the Target window or the Build window if you are still pressing CTRL.*











## Commands on each toolbar

The commands on the toolbars provide access to the main debugging functions. You can also use the Keyboard shortcuts for most debugging operations, and Access keys support all operations; see Appendix A.

### Breakpoint toolbar



*Commands on the Breakpoint toolbar*

To issue this command:	Click:	Press:
Toggle Breakpoint		F5
Enable Breakpoint		none available
Disable Breakpoint		none available
Configure Breakpoint(s)		CTRL+F5
Reset All Breakpoints		ALT+F5
Enable All Breakpoints		CTRL+SHIFT+F5
Disable All Breakpoints		CTRL+ALT+F5
Remove All Breakpoints		SHIFT+F5

## Using and configuring the interface




---

### Debug toolbar

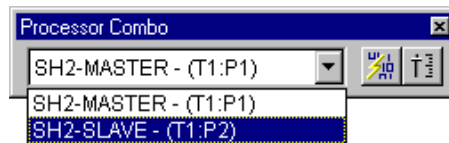


*Commands on the Debug toolbar*

To issue this command:	Click:	Press:
Run all Processor(s)		CTRL+F9
Stop all Processor(s)		none available
Run		F9
Run to Address		SHIFT+F9
Run to Cursor		ALT+F9
Stop		F9
Single Step (into)		F7
Forced Step (into)		none available
Step Over		F8
Step Out		CTRL+F8
Unstep		CTRL+F7
Step Run In		SHIFT+F7
Step Run Out		SHIFT+F8
Step Run		ALT+F7
Step Run Until		ALT+F8



To issue this command:	Click:	Press:
Set Cursor to PC		CTRL+SHIFT+P
Set PC to Cursor		CTRL+ALT+P
Restart		CTRL+SHIFT+R

### Processor Combo toolbar



The Processor Combo toolbar shows the current processor for the current target. The toolbar also provides point and click access for loading program files and configuring the current processor.

#### *Commands on the Processor Combo toolbar*

To issue this command:	Click:	Press:
Load Program File		CTRL+SHIFT+C
Configure Processor		none available

### Input/Output window

The Input / Output window appears automatically and displays the:

- **Build tab** with the specified build utility's output when you build your project.
- **Log tab** with all messages generated by the current target.
- **Scripts tab** with all messages generated by the current script.
- **Workshop tab** if you are running GDWorkshop. The Workshop tab displays GD-M log information, and lets you control the log events.

**NOTE:** For more information on the Input/Output window see page 34.

# Using and configuring the interface

---

## Region toolbar

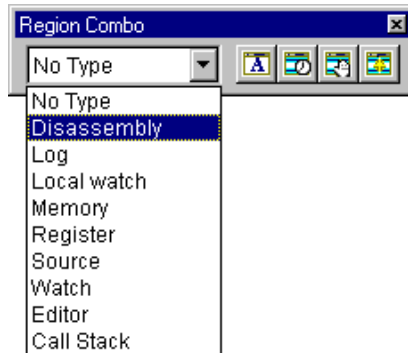


The Region toolbar lets you set or change a region’s type.

*Commands on the Region toolbar*





To create this region:	Click:	Or press:
Disassembly		ALT+1
Local Watch		ALT+3
Memory		ALT+4
Register		ALT+5
Source		ALT+6
Watch		ALT+7
Edit		ALT+8
Call Stack		ALT+9

**NOTE:** To stop the display from updating in all regions press **CTRL+SHIFT+U**.

**Region Combo toolbar**

The Region combo toolbar lets you set the rate at which a region's display updates, and change a region's type.

*Commands on the Region Combo toolbar*

To set this command:	Click:
Region configuration	
Window update rate	
Stop all window updates	
Update all regions	

# Using and configuring the interface

---

## Splitter toolbar



The Splitter toolbar lets you split existing regions to create new regions.











*Commands on the Splitter toolbar*

To issue this command:	Click:	Press:
Split Left		CTRL+SHIFT+LEFT ARROW
Split Right		CTRL+SHIFT+RIGHT ARROW
Split Up		CTRL+SHIFT+UP ARROW
Split Down		CTRL+SHIFT+DOWN ARROW
Delete Region		CTRL+D

**Standard toolbar**

The Standard toolbar provides point and click access for opening a new window, and creating, opening, and saving sessions.

*Commands on the Standard toolbar*

To issue this command:	Click:	Press:
New window		CTRL+N
New Session		CTRL+SHIFT+N
Open Session		CTRL+O
Save Session		CTRL+S
Cut		CTRL+X
Copy		CTRL+C
Paste		CTRL+V
Print		CTRL+P
About Box		none available
Help		F1

# Using and configuring the interface

---

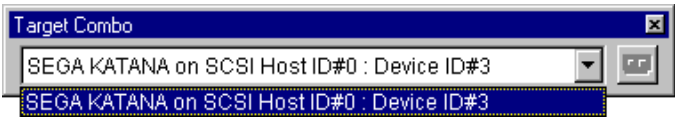
## Target window

The Target window shows all the targets that CodeScape is connected to and the processors available in each target. The processor status for each target is shown in the Target processor display.

When you create a new window, CodeScape uses the target information from the selected processor on the target. The Target window provides point and click access for selecting a target processor.

*NOTE: The Target window can be docked at the top and bottom of the main window, or left free floating.*

## Target Combo toolbar



The Target Combo toolbar shows the current target and lets you to configure it.

### Commands on the Target Combo toolbar












To issue this command:	Click:
Target Configure	

*NOTE: The Serial Setup button only appears if you are connected to a target with a serial port.*



**Editor toolbar**

The Edit toolbar provides point and click access to the editing actions.

To issue this command:	Click:	Press:
Create a new Editor file.		none available
Open an existing Editor file.		none available
Save the current Editor file.		none available
Undo the last action.		CTRL+Z
Redo the last action.		none available
Search for a string.		CTRL+F
Replace the current selection.		none available
Toggle a Book Mark on or off.		none available
Move to the next Book Mark in the file.		none available
Move to the previous Book Mark in the file.		none available
Delete all Book Marks.		none available

# Using and configuring the interface

---

## Workshop toolbar



The Workshop toolbar provides point and click access to the GD Workshop log options.

To issue this command:	Click:
Open Door	
Close Door	
Nudge	
Switch to GD-ROM	
Switch to Emulator	
Hard Errors	

## Status Bar

The status bar is the horizontal area at the bottom of CodeScape’s main window. It provides contextual information about commands on the interface and any other information about the current state of what you are viewing.

To display the status bar:

- Click View then select Status Bar.

# ***How windows and regions work***

---

A **Window** is a frame that you can configure as a Region and split to create multiple Regions.

A **Region** lets you view information about your project.

To view a project's regions simultaneously you can:

- Open and close, minimize and maximize, cascade, and tile multiple windows.
- Split windows into multiple regions to display different types of information such as memory contents and source code.
- Resize windows.
- Resize regions by moving the Splitter bars.
- Proportionally resize a window's regions.

Use the Region configuration dialog box to configure fonts and colors for each region type, each individual region, and each processor. This lets you to differentiate between processors, show associated regions, and represent changes in memory.


***NOTE:** If you use a Windows® 95 or a Windows® 98 machine, creating too many new windows or too many new regions causes CodeScape to run out of system resources.*

## ***How windows and regions work***


---

### **Using windows**


#### **Open a new window**

- Click Window, then click New window.  
-OR-
- Click  on the Standard toolbar.  
-OR-
- Press CTRL+N.


#### **Minimize a window**

- On the window title bar click  .  
-OR-
- On the System menu, click Minimize.

#### **Maximize a window**

- On the window title bar click  .  
-OR-
- On the System menu, click Maximize.

#### **Close a window**

- On the window title bar click  .  
-OR-
- On the System menu, click Close.  
-OR-
- Press CTRL+F4.

*NOTE: If you delete the only region in a window, the window is deleted as well.*

#### **Close all windows**

- Click Window, then click Close all Windows.

### **Move a window**

- 1) Click the window's title bar.
- 2) Drag the window to the required position.

### **Move between windows**

- Press CTRL+TAB.

### **Resize a window**

- 1) Point to the window boarder.
- 2) Click and drag the window outline to the required size.

To proportionally resize a region in a window:

- 1) Click Window, then select Proportional resizing.
- 2) Point to the window's border.
- 3) Click and drag the window to the required size.

### **Load the current session when CodeScape restarts**

- Click Window, then click to select Load last session on startup.

### **Cascade all windows**

- Click Window, then click Cascade.

### **Tile all windows**

- Click Window, then click Tile.

### **Arrange Icons**

To arrange all minimized region windows at the bottom of the session window:

- Click Window, then click Arrange Icons.

### **Using regions**

#### **Change a region's type**

To change a region's type:

- Click Region, then point to Type, then click a region type.  
-OR-
- On the Region Combo box, select a region type from the drop down list.  
-OR-
- Click a Region Type icon on the Region toolbar.  
-OR-
- CTRL+Right-click, then click a region type.

#### **Navigate a region**

To scroll through a region:

- Use the LEFT ARROW and RIGHT ARROW keys to move the cursor a single character at a time.
- Use the UP ARROW and DOWN ARROW keys to move the cursor up and down a line at a time.
- Use PAGE UP and PAGE DOWN to move up and down a page at a time.
- Use the HOME and END keys to move to the first visible line and the last visible line of a file.

To move through a region's fields:

- To move the cursor to the next field, press TAB.
- To move the cursor to the previous field, press SHIFT+TAB.

*NOTE: At the end of a field the cursor moves to the next field;  
at the end of the last field the cursor moves to the next line.*

### **Move between regions**

To move to the region to the left:

- Click anywhere in the region to the left.
- OR-
- Press CTRL+LEFT ARROW.

To move to the region to the right:

- Click anywhere in the region to the right.
- OR-
- Press CTRL+RIGHT ARROW.

To move to the region above:

- Click anywhere in the region above.
- OR-
- Press CTRL+UP ARROW.

To move to the region below:

- Click anywhere in the region below.
- OR-
- Press CTRL+DOWN ARROW.

### **Create new regions**

To create a new region:


- Open a new window (CTRL+N).
- OR-
- Split an existing region.

## ***How windows and regions work***


---

### **Split regions**


To split a region to the left:

- Click Region, point to Split, then click Left.  
-OR-
- Click  on the Splitter toolbar.  
-OR-
- Press CTRL+SHIFT+LEFT ARROW.


To split a region to the right:

- Click Region, point to Split, then click Right.  
-OR-
- Click  on the Splitter toolbar.  
-OR-
- Press CTRL+SHIFT+RIGHT ARROW.

To split a region above:

- Click Region, point to Split, then click Up.  
-OR-
- Click  on the Splitter toolbar.  
-OR-
- Press CTRL+SHIFT+UP ARROW.


To split a region below:

- Click Region, point to Split, then click Down.  
-OR-
- Click  on the Splitter toolbar.  
-OR-
- Press CTRL+SHIFT+DOWN ARROW.



### Delete a region

Make the region active, then do one of the following:

- Click Region, click Delete.  
-OR-
- Click  on the Splitter toolbar.  
-OR-
- In the region, press CTRL+Right-click and click Delete Region.  
-OR-
- Press CTRL+D.

*NOTE: If there is only one region in a window, deleting it deletes the window as well.*

### Update the display in all open regions


- Click Region, then click Update all regions now.  
-OR-
- Press CTRL+U.

**Configuring regions**

**Region Configuration dialog box**

In the Region Configuration dialog box are tab commands for configuring fonts and colors, and setting the update rate for a single region, a region type, and each processor.

To Configure an active region:

- Right-click, click Properties...  
-OR-
- On an active region’s title bar, double-click .

The Region Configuration dialog box appears. Do the following:

- 1) Set the Mode and specify any commands in the *Target processor* text box.
- 2) Specify any options in the Target, Processor, or Region type lists.
- 3) Then do one of the following:
  - Click Apply to view your configuration changes without leaving the dialog box.
  - OR-
  - Click OK to set the configuration changes for your project.

*Using the Region Configuration dialog box*

To configure:	Select:
The currently active region in a project.	Apply to active region only.
All regions of a selected type on all processors.	Apply to all regions of selected type. Then select a Region type.
A specific Target Processor, and Region type.	Apply to all regions of the selected target. Then select a Target Processor, and a Region type.

## **Set the color and font**

Use the Color and Font tab commands to differentiate between processors, show associated regions, and represent changes in memory.

To set the color attributes for the specified Mode:

- 1) Select the Color tab.
- 2) Select the attribute whose color you want to change.
- 3) Set the region Foreground color.
- 4) Set the region Background color.
- 5) Click OK.

To set the font type and size for the specified Mode:

- 1) Select the Font tab.
- 2) Click Change font.
- 3) Specify the region Font, Font style, and Size.
- 4) Set the Effects you require.
- 5) Click OK.

## **Set the region update rates**

Use the Update Rate tab to specify when CodeScape will update information in each region.

If the update rate for a region's display interrupts the target causing jitter in your program, set the Foreground and Background sliders to Min.

To specify when CodeScape will update information in a region:

- 1) Select the Update tab.
- 2) Drag the Foreground slider to set the update rate for when the region has focus. Set the slider to Max to continually update the display (approximately 14Hz). Set the slider to Min to update the display at approximately 1/10th of the Max setting.
- 3) Drag the Background slider to set the update rate for when the region does not have focus. Set the slider to Max to continually update the display. Set the slider to Min to prevent updates to the display.

## How windows and regions work

---

### Target window

The Target window can be docked at the top and bottom of the main window, or left free floating.

When you run CodeScape it scans for valid targets and displays them in the Target window. The Target window shows all the targets that CodeScape is connected to and the processors available in each target. The processor status for each target is shown in the Target processor display.

When you create a new window, CodeScape uses the target information from the selected processor on the target.


### Using the shortcut menu on the Target window

#### *Controlling target processor execution*


Right-click, then click:	To:
Configure Processor...	Set the update rate for the current processor.
Simulate Processor	Run the Simulator.
Execution	Run, stop, and restart your program. Run your program until it executes a specified address. Run all of your program files simultaneously. Stop all of your programs running simultaneously. Use the single stepping commands, or run the step commands.
Breakpoints	Add, enable, disable, configure, reset, or remove data breakpoints.
Reset Target	Perform a soft reset or a hard reset. If you reset the target you are prompted to reload the Program File.
Load Program File	Download a program file to the selected processor on the target.
Allow Docking	Toggle docking of the window on or off.
Hide	Hide the window.

### **Target Processor display**

To show the processor(s) for a target:

- Double-click on the target status line.  
-OR-
- Click  .

To hide the processor(s) for a target:

- Double-click on the target display line.  
-OR-
- Click  .

### **Input/Output window**

The Input / Output window appears automatically and displays the:

- **Build tab** with the specified build utility's output when you build your project.  
Any standard format errors and warnings are shown in the Build tab. You can scroll through the information as it is generated, or press F4 to move through any listed errors one at a time. If you use:
  - CodeScape's Edit region it automatically opens your project file at the line containing the first error or warning. You can then use the Build tab to navigate to all subsequent errors. If there is no active Edit region CodeScape creates one for you.
  - An external editor, double-click an entry in the Build tab to invoke the editor and open the source file at the line containing the error or warning. Some external editors do not support this option and will open without displaying the line at which the error occurred.
- **Log tab** with all messages generated by the current target.  
For example, you can use `printf ()` in your code to output a message to the Log region when a breakpoint triggers.
- **Scripts tab** with all messages generated by the current script.
- **Workshop tab** if you are running GDWorkshop. The Workshop tab displays GD-M log information, and lets you control the log events.

*NOTE: You can dock the Input/Output window at the top and bottom of the main window, or leave it free floating.*

*NOTE: If you enable high level optimization when you build your project the compiler output can make source-level tracing confusing.*

*NOTE: Text strings longer than 132 characters are truncated when displayed in the Log tab.*

## Using the shortcut menu on the Build tab

*Configure, make, then build a project*

Right-click, then click:	To:
Setup Project...	Specify file locations for making a project current and building it.
Setup Editor...	Specify the editor that you want to use for your project.
Make	Make your project current by building it.
Next Error	Move to the next error in the list.
Clear	Clear the contents of the Project Build window.
Allow Docking	Toggle docking for the window on or off.
Hide	Hide the window.

## Using the shortcut menu on the Log tab

Right-click, then click:	To:
Configure Log...	Configure the Log tab.
Print...	Print the contents of the Log tab.
Save To File...	Save the contents of the Log tab to a file.
Execution	Run, stop, and restart your program. Run your program to the cursor position, or until it executes a specified address. Run all of your program files simultaneously. Stop all of your programs running simultaneously. Use the single stepping commands, or run the step commands.
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.
Reset Log	Clear the contents of the Log tab.

## ***How windows and regions work***

---

### **Using the shortcut menu on the Scripts tab**

<b>Right-click, then click:</b>	<b>To:</b>
Run Script	Select and run a script.
Clear	Clear the contents of the Script tab.
User Scripts	This option appears in gray until you add a script to the menu. When you add a script its name appears on the menu.
Allow Docking	Toggle docking for the window on or off.
Hide	Hide the window.

### **Using the shortcut menu on the Workshop tab**

<b>Right-click, then click:</b>	<b>To:</b>
Disable Updates	Disable Workshop message logging.
Close Door	Close the door.
Switch To Emulator / Switch To GD-ROM	Toggle between the emulated GD-ROM image and the actual GD-ROM.
Nudge	Create a soft error on the next operation.
Hard Errors On	Enable hard errors as defined in Workshop. If you use this command, enable it before emulating.
Clear	Clear the contents of the Workshop tab.
Allow Docking	Toggle docking for the window on or off.
Hide	Hide the window.



## The Source and Disassembly regions

The Source and Disassembly regions let you debug your program code from different views.

- In a Disassembly region are commands for debugging your program at instruction level (assembly code).
- In a Source region are commands for debugging your original source code.

When you edit your source code the changes are displayed in the corresponding Source region when the display is updated. A \* appears in a Source region's title bar if you edit your source code and do not re-build the program file. Always save any changes that you make to a file edited in an external editor before using the Make option to compile and build your project in CodeScape.

*NOTE: Place the mouse pointer over a variable or expression to quickly view its' value.*

*NOTE: Before you edit a program file from a UNIX target, convert it to a DOS readable format using a utility such as `to_dos` (use `to_unix` to return the file to a UNIX format).*

*NOTE: If no debug information appears in a Source region, compile all source files for your project with debugging turned on.*

If no source is available you can tell CodeScape to show the disassembly instead. To do this:

- 1) Click Tools, then click Options...  
The Options dialog box appears.
- 2) Select the Automatic Source/Disassembly Switching check box.
- 3) Click OK.

## ***How windows and regions work***

---

### **Using the shortcut menu in a Source region**


The commands on the shortcut menu are for debugging in the region and configuring the source view. Right-click anywhere in the region to access the shortcut menu.

### **Copy in the Source region**

- 1) In the Source region, select the text you want to copy by highlighting it.
- 2) Right-click, then click Copy.

The selection is copied, then pasted to the clipboard.

### **Lock the display origin to an expression**

- 1) On the Source region title bar click .  
The Goto Address... dialog box appears.
- 2) Enter an expression for the region origin.
- 3) Click OK.

#### *Configuring the Source view*

<b>Right-click, then click:</b>	<b>To show the:</b>
Show Address	Corresponding address for the first line of code generated by the source code line.
-OR-	
Show Line Nos.	Line numbers for each line of source in the left-hand column.



#### *Accessing the debugging commands in a Source region*

<b>Right-click, then select:</b>	<b>To click commands to:</b>
Execution	Run, stop, and restart your program. Run your program to the cursor position, or until it executes a specified address. Run all of your program files simultaneously. Stop all of your programs running simultaneously. Use the single stepping commands, or run the step commands.
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.

*Setting the cursor and the display in a Source region*

Right-click, then click:	To:
Set Cursor to PC	Show the source code from the value of the PC.
Set PC to Cursor	Change the PC at the current cursor position.
Goto Address...	Enter an expression for the region origin to go to.
Goto Source File...	Select the required source file.
View As	The program file in the active region as source code, or assembler code, or both source and assembler code.
Tools	Search in the Source region. Find the next item in the search.
Tab Width...	Enter a value to set the tab size in spaces.
Properties	Configure fonts and colors. Set the update rate for a single region, a region type, and each processor. Change the tab settings.

**Synchronize the cursors in a Source and Disassembly region**

- 1) In the Source region:
  - Right-click and click Synchronize Cursor.
  - OR-
  - On the Source region title bar click .
- 2) In the Disassembly region:
  - Right-click and click Synchronize Cursor.
  - OR-
  - On the Disassembly region title bar click .

The cursors for the Disassembly and Source regions are now synchronized. When you move the cursor in the region with focus, the cursor in the synchronized region shows the corresponding line of code.

**NOTE:** *You can only synchronize regions that are in the same window and are connected to the same target processor.*

## ***How windows and regions work***

---

### **Goto an address**

- 1) Do one of the following:
  - Click Edit, then click Go To (CTRL+G).
  - OR-
  - Right-click, click Goto Address.

The Goto Address dialog box appears. (This dialog box works in the same way as the Expression Evaluator.)

- 2) Enter an expression for the address to go to.
- 3) Click OK.

### **Go to a source file referenced in the program file**

- 1) Right-click and click Goto Source File.  
The List Files in Program File dialog box appears.
- 2) Select the required source file.
- 3) Click OK.

If the path is incorrect an error message appears in the Source region. Click Project, then click Edit Source Path and enter the correct path for the source files.

*NOTE: Code is not generated for data-only files, or if the -g command is not set when compiling. If code is not generated an error message appears.*

### **Evaluate a specific expression**

- 1) Select an expression in the region.
- 2) Right-click, click Evaluate...

### **Change the tab settings**

- 1) Right-click, point to Properties then click Tab Width...  
The Change Tab Size dialog box appears.
- 2) Enter a value for the number of spaces used to represent a tab.
- 3) Click OK.

### Search in the Source region

- 1) Right-click in the Source region, click Tools, then click Find.
- 2) Type the Search string in the *Find what* text box.
- 3) To search for whole words and not parts of a larger word, select the *Match whole word only* check box.
- 4) If the search is case sensitive, select the *Match case* check box.
- 5) Click OK.

The search will start from the current cursor position and continue until the end of the file.

**NOTE:** *Right-click, then click Find next to continue searching for the same item.*

## ***How windows and regions work***

---

### **Using the shortcut menu in a Disassembly region**

The commands on the shortcut menu are for debugging in the region and configuring the disassembly view. Right-click anywhere in the region to access the shortcut menu.


### **Copy in the Disassembly region**


- 1) In the Disassembly region, select the text you want to copy by highlighting it.
- 2) Right-click, then click Copy.

The selection is copied, then pasted to the clipboard.

### **Lock the Disassembly region**

You can lock the view origin to the PC, a register, or a memory location.

- 1) On the Disassembly region title bar click . The Goto Address dialog box appears.
- 2) In the *Expression* text box, enter a valid expression:
  - Value of the PC to lock the view to the PC. Click OK.
  - OR-
  - Name of the register to lock the view to a register. Click OK.
  - OR-
  - In the *Expression* text box, enter the address of the memory location to lock the view to a memory location. Click OK.

**NOTE:** To unlock the view origin, click  again.

*Configuring the disassembly view*

Right-click, then click:	To show the:
Show Address	Location address of the disassembled code.
Show Labels	Symbolic label replacement of the disassembled code.
Show Opcode Words	Op-code in words for the disassembled region.
Show Hexadecimal	Operand values in hexadecimal.
Show Uppercase	Instructions in upper case.
Show Symbols	Operand values as symbols.
Show EAs & Lits	Effective address and literals.

*Accessing the debugging commands in a Disassembly region*

Right-click, then select:	To:
Execution	Run, stop, and restart your program. Run your program to the cursor position, or until it executes a specified address. Run all of your program files simultaneously. Stop all of your programs running simultaneously. Use the single stepping commands, or run the step commands.
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.



## How windows and regions work

---

### Setting the cursor and the display in a Disassembly region

Right-click, then click:	To:
Set Cursor to PC	Show the source code from the value of the PC.
Set PC to Cursor	Change the PC at the current cursor position.
Goto Address...	Enter an expression for the region origin to go to.
Tools	Search in the Disassembly region. Repeat the last search run. Specify an address to disassemble to a file.
Properties	Configure fonts and colors. Set the update rate for a single region, a region type, and each processor. Change the tab settings.

### Synchronize the cursors in a Disassembly and Source region

- 1) In the Disassembly region:
  - Right-click and click Synchronize Cursor.
  - OR-
  - On the Disassembly region title bar click .
- 2) In the Source region:
  - Right-click and click Synchronize Cursor.
  - OR-
  - On the Source region title bar click .

The cursors for the Disassembly and Source regions are now synchronized. When you move the cursor in the region with focus, the cursor in the synchronized region will show the corresponding line of code.

**NOTE:** You can only synchronize regions that are in the same window and are connected to the same target processor.



### Goto an address

- 1) Do one of the following:
  - Click Edit, then click Go To (CTRL+G).
  - OR-
  - Right-click, click Goto Address.The Goto Address dialog box appears.
- 2) Enter an expression for the address to go to.
- 3) Click OK.

### Evaluate a specific expression

- 1) Select an expression in the region.
- 2) Right-click, click Evaluate...

### Search in the Disassembly region

- 1) Right-click in the Disassembly region, point to Tools, then click Find.
- 2) Type the Search string in the *What am I searching for:* text box.
- 3) Type the Start address in the *Search from:* text box.
- 4) If the search is case sensitive, select the *Case sensitive* check box.
- 5) Select one of the following radio buttons:
  - Length (the amount of data).
  - OR-
  - End Address.
- 6) Type the search item in the text box below.
- 7) Select one of the following radio buttons: All fields (default), Words, Opcode, OpSrc, OpDest, or Label (address).
- 8) Click OK.

**NOTE:** *Right-click then click Find next to continue searching for the same item.*

## ***How windows and regions work***

---

### **Specify an address to disassemble to a file**

This general purpose dialog box is for writing a block of memory or disassembly in hexadecimal to a file.

- 1) Right-click in the Source region, point to Tools, then click Disassemble to File.
- 2) In the *Destination Filename* text box, enter the name of the file to write to.
- 3) In the *Start Address* text box, enter the start address in hexadecimal.
- 4) Do one of the following:
  - Select Length and enter the length in hexadecimal.
  - OR-
  - Select End Address and enter the end address in hexadecimal.
- 5) Click OK.

## The Call Stack region

Use the Call Stack region to view a list of active function calls. Viewing the Call Stack can help you trace the course of function execution. When the target stops, for example if a breakpoint occurs, CodeScape displays the name, label, or address of the current function at the top of the list in the Call Stack region. Execution trace history is shown below the current function with its start point at the bottom of the list.

To navigate to a specific function call in active Source, Disassembly, Watch, and Local Watch regions, in the Call Stack region, double click on a function. CodeScape highlights the function as it occurs in the active regions.

### Using the shortcut menu in a Call Stack region

Right-click, then select:	To click commands to:
Show Parameter Names	Toggle the function parameter names on or off.
Show Parameter Types	Toggle function parameter types on or off.
Show Parameter Values	Toggle function parameter values on or off.
Show Parameter Registers	Toggle function parameter registers on or off.
Show Octal	Display function values in octal.
Show Decimal	Display function values in decimal.
Show Hexadecimal	Display function values in hexadecimal.
Execution	Run, stop, and restart your program. Run your program until it executes a specified address. Run all of your program files simultaneously. Stop all of your programs running simultaneously. Use the single stepping commands, or run the step commands.
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.
Properties	Configure fonts and colors and set the region and processor update rates.

**NOTE:** Use *Run to Cursor* to return to a specific function outside of the active one.

### **The Watch and Local Watch regions**

The Watch and Local Watch regions display variables and expressions, one per row.

Each row has four columns, the expression appears in the third column and its value (if applicable) appears in the fourth column. If the:

- First column contains a ‘.’ you can place a watch point on the expression.
- Second column contains a ‘+’ you can expand the expression.
- Second column contains a ‘-’ you can collapse the expression.

In a Watch or Local Watch region, you can:

- Highlight changes in data values between execution steps.
- Edit the value of an expression in the Watch region and the Local Watch region.

*NOTE: You can only edit the actual expression in a Watch region.*

### **C++ name demangling in a Watch or Local Watch region**

C++ name de-mangling is performed on all variable names. This means that you can enter the symbol for a name as it appears in your original source.

You can browse data to:

- Expand and collapse branches of the hierarchical view of the structure.
- See exactly where the structures are in memory.
- Edit the values of any variables.

All C types are supported including:

- structs
- unions
- arrays
- enumeration (enum)
- floats / double

*NOTE: If you place a watch (data) breakpoint on a member of a union it will trigger for all members of that size, regardless of type. This also applies to anonymous unions, except that two members of the same size appear as two variables sharing the same address in memory.*

### **Expanding expressions**

When you expand an expression, each child expression is indented and shown directly below the parent.

For example:

```
parent
  child
  child
```

Expressions are added to expanded:

- Pointers, to show the dereferenced item.
- Arrays. An expression is added for each element of the array.
- Structures. An expression is added for each member.

### The Watch region

In the Watch region you can enter variables and expressions. The scope of variables in a Watch region is global. If an expression goes out of scope during program execution, a message appears.

- If an expression's value can be determined, as is the case for a static variable, its value is shown in the region.
- If an expression's value cannot be determined, no value is shown.
- If an expression comes back into scope, its value is shown.

### Using the shortcut menu in a Watch region

Right-click, then select:	To click commands to:
Cut	Cut the current selection in the Editor file and paste it to the clipboard.
Copy	Copy the current selection in the Editor file and paste it to the clipboard.
Paste	Insert the contents of the clipboard at the current cursor position.
Delete	Delete part of a structure.
Open	Expand a structure or array.
Close	Collapse a structure or array.
Insert	Insert a new watch expression.
Append	Add a variable to the end of the active list.
Show Headers	Toggle the header bar on or off.
Keep in View	Keep the cursor in view if it is possible.
Show Octal	Display watch expressions in octal.
Show Decimal	Display watch expressions in decimal.
Show Hexadecimal	Display watch expressions in hexadecimal.
Edit Watch Value...	Modify the value of a variable or watch expression.

Right-click, then select:	To click commands to:
Execution	Run, stop, and restart your program. Run your program until it executes a specified address. Run all of your program files simultaneously. Stop all of your programs running simultaneously. Use the single stepping commands, or run the step commands.
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.
Highlight Changes	See where in memory an expression changed.
Cache Expanded Symbols	Save the state of an expanded function to the session file. The next time that the function is accessed it will automatically expand to its saved state.
Properties	Configure fonts and colors. Set the update rate for a single region, a region type, and each processor.

## ***How windows and regions work***

---

### **Browsing data in a Watch region**

#### ***Add a symbol or variable***

- Right-click, click Insert to add a symbol or variable at the current cursor position.  
-OR-
- Right-click, click Append to add symbol or a variable at the end of the current list of variables.  
-OR-
- Press return to enter a new symbol or variable at the current cursor position.

#### ***Expand a structure or array***

Select the structure or array you want to expand, then:

- Click on '+'.  
-OR-
- Press SPACEBAR.  
-OR-
- Right-click and click Open/Close.

#### ***Collapse a structure or array***

Select the structure or array you want to collapse, then:

- Click on '-'.  
-OR-
- Press SPACEBAR.  
-OR-
- Right-click and click Open/Close.



## **Editing variables in a Watch region**

### ***Modify the value of a variable or watch expression***

- Select the value to be changed. Press ENTER.
- OR-
- Press CTRL+ALT+E.

The Expression Evaluator dialog box appears. Enter a valid expression. Click OK.

### ***Edit a variable's data value (structure, array or union)***

- 1) Double-click the value to be edited.
- 2) Edit the value.
- 3) Do one of the following:
  - Press ENTER.
  - OR-
  - Press CTRL+ALT+E to display the Expression Evaluator dialog box.

### **Delete a parent expression and all child expressions**

- 1) Expand the structure or array.
- 2) Select the component you want to delete, then:
  - Right click and click Delete.
  - OR-
  - Press DELETE.

*NOTE: If you delete a parent expression, any children are also removed from the region.*

### ***Delete a parent expression and move all children up one level***

- 1) Expand the structure or array.
- 2) Select the component you want to delete.
- 3) Press SHIFT+DELETE.

## How windows and regions work

---

### The Local Watch region

The Local Watch region automatically displays all local variables in view from the current position of the PC (program counter). Variables are automatically added to the display as they come into the scope of a function.

**NOTE:** *If there is more than one variable of the same name in the current scope, all but the inner most variable of that name are unavailable and are shown in gray.*

**NOTE:** *If there is more than one variable of the same name in the same scope they are shown in italics.*

#### Using the shortcut menu in a Local Watch region

Right-click, then select:	To click commands to:
Copy	Copy the current selection in the Editor file and paste it to the clipboard.
Delete	Delete the current selection.
Open	Expand a structure or array.
Close	Collapse a structure or array.
Show Headers	Toggle the header bar on or off.
Keep in View	Keep the cursor in view if possible.
Show Octal	Display local watch expressions in octal.
Show Decimal	Display local watch expressions in decimal.
Show Hexadecimal	Display local watch expressions in hexadecimal.
Edit Local Value	Modify the value of a variable or watch expression.
Execution	Run, stop, and restart your program. Run your program until it executes a specified address. Run all of your program files simultaneously. Stop all of your programs running simultaneously. Use the single stepping commands, or run the step commands.

Right-click, then select:	To click commands to:
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.
Highlight Changes	See where in memory an expression changed.
Cache Expanded Symbols	Save the state of an expanded function to the session file. The next time that the function is accessed it will automatically expand to its saved state.
Properties	Configure fonts and colors. Set the update rate for a single region, a region type, and each processor.

## Browsing data in Local Watch region

### ***Expand a structure or array***

Select the structure or array you want to expand, then:

- Click on '+'.  
-OR-
- Press SPACEBAR.  
-OR-
- Right-click and click Open/Close.

### ***Collapse a structure or array***

Select the structure or array you want to collapse, then:

- Click on '-'.  
-OR-
- Press SPACEBAR.  
-OR-
- Right-click and click Open/Close.

## ***How windows and regions work***

---

### **Editing variables in a Local Watch region**

#### ***Modify the value of a variable or watch expression***

- 1) Do one of the following:
  - Select the value to be changed. Press ENTER.
  - OR-
  - Press CTRL+ALT+E.The Expression Evaluator dialog box appears.
- 2) Enter a valid expression.
- 3) Click OK.

#### ***Delete a parent expression and all children***

- 1) Expand the structure or array.
- 2) Select the component you want to delete, then:
  - Right click and click Delete.
  - OR-
  - Press DELETE.

***NOTE:*** *If you delete a parent expression, any child expressions are also removed from the region.*

#### ***Delete a parent expression and move all children up one level***

- 1) Expand the structure or array.
- 2) Select the component you want to delete.
- 3) Press SHIFT+DELETE.

## The Memory region

Use the Memory region to view the targets memory contents from a specific address. In a Memory region you can view memory as ASCII characters, Bytes, Words, or Longs. Write protect an area of memory to prevent memory contents changing in the current memory region.

As you scroll through a Memory window the cursor moves a line at a time and the slider speed increases. The slider automatically returns to the center position when you stop scrolling. Double-click a variable or expression to quickly view and edit its value.

### Using the shortcut menu in a Memory region

Right-click, then select:	To click commands to:
Display Bytes	Display memory as bytes.
Display Words	Display memory as words.
Display Longs	Display memory as longs.
Display Quadwords	Display memory as quadwords.
Display ASCII	Display the ASCII value for each byte memory.
Highlight Changes	See where the target's memory changed.
Set Bytes Per Line...	Display a specific number of bytes per line.
Edit ASCII	Change an ASCII value in the Memory region.
Edit Memory Value	Change a value in the Memory region.
Follow Pointer	Follow a pointer in memory.
Goto Address...	Set the origin.
Write Protect	Toggle write protect.
Execution	Run, stop, and restart your program. Run your program to the cursor position, or until it executes a specified address. Run all of your program files simultaneously. Stop all of your programs running simultaneously. Use the single stepping commands, or run the step commands.
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.

## How windows and regions work


---

Right-click, then select:	To click commands to:
Tools	Search for a pattern in memory. Repeat the last search. Fill a range of memory with data. Write a block of memory in hexadecimal to a file.
Properties	Configure fonts and colors. Set the update rate for a single region, a region type, and each processor.

### Viewing Memory

#### View Memory regions

Do one of the following:

- Click Region, point to Type and click Memory.  
-OR-
- On the Region toolbar, click .
- OR-
- In any region, CTRL+Right-click and click Memory.

#### Set the origin

- 1) Right-click and click Goto Address...  
The Goto Address... dialog box appears.
- 2) Enter an address or symbol for the new origin.
- 3) If you enter an invalid symbol a warning appears with an command to invoke the Origin dialog box.
- 4) Click OK.

**NOTE:** *The origin is initially set to the value of the PC. You can also set the origin to an expression.*

***Always display memory from a specified address***

- 1) Do one of the following:
  - Click Edit, then click Go To... (CTRL+G).
  - OR-
  - Right-click and click Goto Address...The Goto Address... dialog box appears.
- 2) Type or select a memory location or expression from the Expression list.
- 3) Select Lock, click OK.

***Follow a pointer in memory***

- 1) Select the memory location holding the value of the pointer.
- 2) Right-click and click Follow Pointer (CTRL+T).

The Memory region origin changes to display memory from the location specified by the value of the pointer.

***Write a block of memory in hexadecimal to file***

- 1) In the Memory region, right click, point to Tools and then click Hex Dump to File. The Hex Dump to File dialog box appears.
- 2) In the *Destination Filename* text box, enter the name of the file to write to. In the *Start Address* text box, enter the start address in hexadecimal.
- 3) Then do one of the following:
  - Select Length and enter the length of the memory block in hexadecimal.
  - OR-
  - Select End Address and enter the end address in hexadecimal.
- 4) Click OK.

The specified block of memory is written to a file.

## ***How windows and regions work***

---

### **Editing memory**

#### **Change values in the Memory region**

- 1) In a Memory region, do one of the following:
  - Use the + and - keys to increment or decrement the current value.  
-OR-
  - Double-click or press ENTER, then type over the existing byte, word, or long value.  
-OR-
  - Right-click, click Edit memory value...  
-OR-
  - Press CTRL+ALT+E.The Expression Evaluator dialog box appears.
- 2) Enter a valid expression.
- 3) Click OK.

**NOTE:** *Make sure you enter valid values for the current radix. CodeScape displays all Memory values in hexadecimal.*

#### ***Filling memory with specific data***

To fill a range of memory with a specific data:

- 1) In a Memory region, right-click and select Tools..., click Fill...
- 2) Enter a value in the *Fill Expression* text box.
- 3) Enter a value in the *Start Address* text box.
- 4) Select End address, or Length.
- 5) Enter a value in the text box below.
- 6) Select the Mode as Text (ASCII), Byte, Word, Long, or Quad.
- 7) Do one of the following:
  - Select *Convert Native Endian*, to show the real memory value.  
-OR-
  - Deselect *Convert Native Endian*, to store memory as byte sequences.
- 8) Click OK.



## Searching memory

To define and search an area of memory for a specified pattern of data:

- 1) In a Memory region, right-click and select Tools...
- 2) Click Find. The Find In Memory dialog box appears.
- 3) Enter a search string in the *Find Pattern* text box.  
In Binary, Octal, Decimal, and Hex modes the search pattern is delimited by either commas or semi-colons (optional).
- 4) Enter a value in the *Start Address* text box.  
The default value is the start address of the region. If you have not run a search, the address at the start of the current memory block is used.
- 5) Select End Address, or Length.  
The default value for the end address is the last displayed byte in the Memory region.
- 6) Enter a value in the text box below.
- 7) Select the Width as either Byte, Word, or Long.
- 8) Select Forward or Reverse to specify the direction of the search.
- 9) Click OK.

**NOTE:** *If a specified search is not valid, 'Invalid Address' appears in the field(s) that require editing.*

**NOTE:** *If a match is found its address appears. A search skip over any sensitive areas such as invalid memory areas, write-only memory, and memory reserved for the monitors.*

## How windows and regions work

---

### Width

Width aligns the search pattern with the data in the target's memory. This specifies how the search pattern and the memory contents are compared. The allowable width depends on the search mode selected.

*Valid mode and width combinations*

For this mode:	Valid widths are:
Binary	Binary, Word, Long
Decimal	Byte
Hex	Binary, Word, Long
Text	N/A

**These patterns are equivalent with Hex and Byte widths set:**

```
FF,FF,FF,FF,34,DC
FF;FF;FF;FF;34;DC
FFFFFFFF34DC
```

### The \ specifier

Use the \ specifier to include special characters in text searches.

**NOTE:** *Always enclose a text search string in quotes.*

### Example

The pattern "How are you\?" searches for "How are you?"

### **The ? wildcard**

The wild card character ‘?’ can be used in Binary and Hex modes. Use ‘?’ to specify a nibble in Hex mode and a bit in Binary mode that always results in a successful match.

#### **Example**

In Hex mode:

FF?F matches FF0F,FF1F,FF2F,...,FFFF

In Binary mode:

????1111 matches 00001111,00011111,...,11111111

### **The @ wildcard**

The wild card character, ‘@’, can be used in Text modes. Use ‘@’ to specify a double-byte character.

#### **Automatic padding**

The search pattern is automatically left-padded for the Binary, Decimal, and Hex modes. The padding type is either ‘0’ or ‘?’ depending on the delimiter used.

## ***How windows and regions work***

---

### **Delimit with commas**

Delimit a search pattern with commas (the default) to left-pad it with zeros. For example, in Hex mode with Byte width:

FFFFFFFF34DC and FF,FF,FF,FF,34,DC do the same search.

The comma separator in is implied in the first search pattern. More examples, in Hex mode and Word width, are:

f,87d,a automatically pads to 000F,087D, 000A

f87da automatically pads to 000F,87DA

, automatically pads to 0000

*NOTE: A single comma used on its own produces the pattern 0000. Use this feature carefully. For example, ;7 automatically pads to 0000,0007*

### **Delimit with semi-colons**

Delimit a search pattern with semi-colons to pad it with the '?' wild card. Examples, in Hex mode and Word width, are:

f;8d;a automatically pads to ???F,?87D,???A

f;87da automatically pads to ???F,87DA

; automatically pads to ????

### **Equivalent search patterns**

Use the comma and semi-colon delimiters to do the same search pattern in different ways. The following patterns are the same when the Hex and Byte widths are set:

```
FF,FF,FF,FF,34,DC
FF;FF;FF;FF;34;DC
FFFFFFFF34DC
```

You can mix the comma and semi-colon delimiters to produce precise search patterns. For example, in Hex mode and Long width: f;f0f0f0f0,ffffff?,7; pads to:  
???????F,F0F0F0F0,FFFFFFF?,???????7

## The Register region

The Register region shows the contents of a processor's register block and flags. Double-click a variable or expression to quickly view and edit its value.

### Using the shortcut menu in a Register region

Right-click, then select:	To click commands to:
Increment Register	Apply the current Increment Value (1 is the default) to the contents of the register.
Decrement Register	Apply the current Decrement Value (1 is the default) to the contents of the register.
Change Inc/Dec Value...	Change the Increment/Decrement Value.
Highlight Changes	See where changes occurred during the last operation.
Write Protect	To prevent data from being written to the currently active Register region.
Edit Register	Change the selected register value.
Column Format	Display registers in two, or four columns. Select Auto to tell CodeScape to choose.
Show Banked Registers	Toggle the banked register display on or off.
Show Float Registers	Toggle the floating point register display on or off.
Execution	Run, stop, and restart your program. Run your program to the cursor position, or until it executes a specified address. Run all of your program files simultaneously. Stop all of your programs running simultaneously. Use the single stepping commands, or run the step commands.
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.
Tools	Save the current Register Block. Restore the last saved Register Block.
Properties	Configure fonts and colors. Set the update rate for a single region, a region type, and each processor.

## ***How windows and regions work***

---

### **View the Registers region**

Click Region, point to Type and click Register.

### **Change the display format**

The registers are displayed in the available area by default. You can set the display to two or four columns.

Display registers in two columns

Do one of the following:

- Right-click, point to Column Format, then click 2 Columns.  
-OR-
- Press CTRL+2.

Display registers in four columns

Do one of the following:

- Right-click, point to Column Format, then click 4 Columns.  
-OR-
- Press CTRL+4.

Display registers in the available area

Do one of the following:

- Right-click, point to Column Format, then click Auto Format.  
-OR-
- Press CTRL+0.

## **Edit register values**

Change the value of a register

- 1) Move the insertion point to the register value you want to change.
- 2) Do one of the following:
  - Use + or - to increment or decrement the current value.  
-OR-
  - Type the new value at the insertion point.  
-OR-
  - Double click a register, type a value or expression, press ENTER. The Expression Evaluator dialog box appears.  
-OR-
  - Press CTRL+ALT-E to invoke the Expression Evaluator dialog box.

*NOTE: Any alphanumeric characters are shown in upper case.*

## **Change the Increment/Decrement Value**

- 1) Right-click and click Change Inc/Dec Value...  
The Register Increment/Decrement dialog box appears.
- 2) Type a value for the amount by which to increment or decrement a register. Click OK.

## **Write protect a register**

- 1) Right-click in the region.
- 2) Then do one of the following:
  - If Write protect is not selected, click Write Protect.  
-OR-
  - If Write protect is selected, exit the shortcut menu.

*NOTE: Write protect only stops register values being changed in the region that is write protected. If you create and edit new Register regions, the changes appear in the write protected region.*

## ***How windows and regions work***

---

### **Enter an expression for the instruction at the current PC**

- 1) Double-click the register value, then:
  - Enter the expression. Press ENTER.
  - OR-
  - Right-click and click Edit Register... (CTRL+ALT+E).The Register Evaluation dialog box appears.
- 2) Enter the expression.
- 3) Click OK.

*NOTE: If you enter an invalid expression, the Register Evaluation dialog box appears showing the Invalid Register Expression.*

### **Save the state of the registers**

Right-click, point to Tools, then click Save Register.

*NOTE: The register states for each target processor are saved one at a time and cannot be stacked. The state of the registers is stored internally to CodeScape, not in a file.*

### **Retrieve the state of the registers**

Right-click, point to Tools then click Restore Register.



### SH4 floating-point exceptions

The SH4's floating point exception handler needs software assistance when the V, O, U, or I bits are enabled in the FPSCR.enable field. An FPU exception is raised for many of the floating point op-codes including fadd, fsub, and fmul, regardless of whether an exception occurs.

When any of these bits are set, the target processor can stop with an exception on a floating point instruction, even if you set safe values. For example, fmul r4, r5 where r4=1.5, and r5=1.5.

The Debug Stub (v2.8.0a onwards) analyzes FPU exceptions to find out if they are valid.

- If an Invalid Floating-Point Exception (Invalid FPU) occurs, it is handled by the Debug Stub and a large loss of processor performance is incurred (several hundred clocks).
- If a Valid Floating-Point Exception (Valid FPU) occurs, it is handled by CodeScape and an even greater loss of processor performance is incurred.

When a Valid FPU occurs, CodeScape displays a status message describing the type of exception that caused it in the Target window. The status messages are:

- FPU error (E)
- Invalid operation (V)
- Divide by Zero (Z)
- Overflow (O)
- Underflow (U)
- Inexact (I)

*NOTE: The Debug Stub passes FPU instructions executed in slots to CodeScape. If an unwanted slotted exception occurs, CodeScape traces around it, then resumes running the program.*

*NOTE: It is recommended that the floating-point enable bits are not used when program performance is required.*

*NOTE: For more information about SH4 floating-point exceptions, refer to the SH7091 Programmer's Manual. The manual is on the Dreamcast SDK CD in the SHC\DOC\SH7091 directory.*

## ***How windows and regions work***

---

### **Hitachi target processor register region display**

#### **General registers**

The Register region shows the values of Hitachi's 16 general registers (Rn) numbered R0-R15.

**R0** works as a fixed source register or destination register in some instructions, and as an index register in:

- Indirect indexed register addressing mode.
- Indirect indexed GBR addressing mode.

**R14** works as the frame pointer during debugging.

**R15** works as a hardware stack pointer (SP) during exception processing.

#### **Control registers**

The Register region shows the values of the SR (Status Register), GBR (Global Base Register), and VBR (Vector Base Register).

#### **System registers**

The Register region displays the MACH and MACL (high and low multiply and accumulate registers), PR (Procedure Register), and PC (Program Counter). The MACH and MACL registers store the results of multiply and accumulate operations. The PR stores a return address from a subroutine procedure.

#### ***Changing the value of a status register***

- 1) Move the insertion point to the register value you want to change.
- 2) Do one of the following:
  - Use + or - to increment or decrement the current value.  
-OR-
  - Type the new value at the insertion point.  
-OR-
  - Press CTRL+ALT+E to invoke the Expression Evaluator dialog box.

**NOTE:** *Any alphanumeric characters are shown in upper case.*

**Setting the status register (SR, SSR, or FPSR) flags**

- 1) Move the insertion point to the flag you want to set.
- 2) Then:
  - Press 1 set the current flag.  
The bit's flag appears in upper-case.  
-OR-
  - Press 0 to clear the current flag.  
The bit's flag appears in lower-case.

**NOTE:** Press SPACEBAR to toggle the status registers.

*Flags shown in the Registers region after a Target Processor reset*

This flag:	Represents this value:
T bit	The MOVT, CMP/cond, TAS, TST, BT (BT/S), BF (BF/S), SETT and CLRT instructions use the T bit to show true (1) or false (0). The ADDV/C, SUBV/C, DIV0U/S, DIV1, NEGC, SHAR/L, SHLR/L, ROTR/L and ROTCR/L instructions also use the T bit to show carry/borrow or overflow/underflow.
S bit	Used by the multiply/accumulate instruction.
Bits 2,3 and 10-31 (Reserved bits.)	Always reads as 0 and must be written as 0.
Bits I3-I10	Interrupt mask bits.
M and Q bits	Used by the DIV0U/S and DIV1 instructions.

## ***How windows and regions work***

---

### **SEA and DEA**

The Register region shows the value of the SEA (Source Effective Address) and DEA (Destination Effective Address). The SEA and DEA show the source effective address (read from) on the left-hand side and the contents of that address (write to) on the right-hand side.

### **Highlight recently changed attributes**

Recently changed attributes are shown briefly in red (default).

To use another color to highlight a changed attribute, do the following:

- 1) Click View, then click Properties.
- 2) Specify the Mode.
- 3) Select the Color tab.
- 4) In the Attribute region, select Highlight data changed.
- 5) In the Effects region, select a new Foreground color.
- 6) Click OK.

***NOTE:*** *You cannot highlight changed attributes by setting a different Background color.*

## The Edit region

The default editor is CodeScape's Edit region where you can manage, edit, and print source files.

When you edit your source code the changes are displayed in the corresponding Source region when the display is updated. A \* appears in a Source region's title bar if you edit your source code and do not re-build the program file. Always save any changes that you make to a file edited in an external editor before using the Make option to compile and build your project in CodeScape.

Any standard format errors and warnings are shown in the Project Build window. You can scroll through the information as it is generated, or press F4 to move through any listed errors one at a time. If you use:

- CodeScape's Edit region it automatically opens your project file at the line containing the first error or warning. You can then use the Project Build window to navigate to all subsequent errors. If there is no active Edit region CodeScape creates one for you.
- An external editor, double-click an entry in the Project Build window to invoke the editor and open the source file at the line containing the error or warning. Some external editors do not support this option and will open without displaying the line at which the error occurred.

*NOTE: Before you edit a program file from a UNIX target, convert it to a DOS readable format using a utility such as `to_dos` (use `to_unix` to return the file to a UNIX format).*

## ***How windows and regions work***

---

### **Using the shortcut menu in an Edit region**

<b>Right-click, then select:</b>	<b>To click commands to:</b>
New	Create a new Editor file.
Open...	Open an existing Editor file.
Recent	View a list of up to ten recently used files.
Save	Save the current Editor file.
Save As...	Save the current Editor file with a specific name.
Cut	Cut the current selection in the Editor file and paste it to the clipboard.
Copy	Copy the current selection in the Editor file and paste it to the clipboard.
Paste	Insert the contents of the clipboard at the current cursor position.
Tabs...	Enter a new tab value.
Undo...	Undo the last action.
Redo...	Redo the last action.
Find...	Search for a string.
Replace...	Replace the current selection.
Go To...	Change the line number of the origin address.
Bookmarks	Toggle a Book Mark on or off; move to the next, or previous Book Mark; or delete all Book Marks.
Properties	Configure fonts and colors in the region.
Syntax Highlighting	Turn syntax coloring on or off. Turn case sensitivity on or off. Specify a color for any or all of the following items in your code: keywords, quotes, comments, default text, and the background.

### Opening and saving files

#### Open an existing file

- 1) Right-click and click Open.  
The File Open dialog box appears.
- 2) Select the required file. Click Open.

#### Open a new file

- Right-click and click New.

#### Save a file

- Right-click and click Save.

*NOTE: If you use the save command for an un-named file, the File Save As dialog box appears.*

#### Save a file with a new name

- 1) Right-click and click Save As.  
The File Save As dialog box appears.
- 2) Enter a new name for the file. Click Save.

### **Search and replace**

#### **Perform searches**

- 1) Move the insertion point to where you want to start searching from.
- 2) Do one of the following:
  - Click Edit, then click Find...
  - OR-
  - Right-click, click Find...

The Find dialog box appears.

- 3) In the Find What text box, enter the search string.
- 4) In the Direction field, select Up, or Down.
- 5) Select any of the following options:
  - Match Case to find only strings that match the case of the characters in your search string exactly.
  - Regular expression if you entered a regular expression in the Find What text box.
  - Wrap around search to continue searching after the end of the document has been reached.
- 6) Do one of the following:
  - Click Find Next to continue searching without replacing a found item.
  - OR-
  - Click Mark All to add a bookmark to all lines containing your search string.



## **Search for and replace a string**

- 1) Move the insertion point to where you want to start replacing from.
- 2) Do one of the following:
  - Click Edit, then click Replace...
  - OR-
  - Right-click, click Replace...

The Replace dialog box appears.

- 3) In the Find What text box, enter the search string.
- 4) In the Replace With text box, enter the new string.
- 5) In the Direction field, select Up, or Down.
- 6) Select any of the following options:

Match Case to find only strings that match the case of the characters in your search string exactly.

- Regular expression if you entered a regular expression in the Find What text box.
  - Wrap around search to continue searching after the end of the document has been reached.
- 7) Do one of the following:
    - Click Find Next to continue searching without replacing a found item.
    - OR-
    - Click Replace to replace the first instance of your search string.
    - OR-
    - Click Replace All to replace all instances of your search string.

### **Cutting and pasting text**

#### **Move text**

- 1) Select the text that you want to move by highlighting it.
- 2) Click Edit, then click Cut (CTRL+X).
- 3) Put insertion point where you want to paste the information.
- 4) Click Edit, then click Paste (CTRL+V).

The text is removed from the original location and appears in its new location.

#### **Copy text**

- 1) Select the text you want to copy by highlighting it.
- 2) Click Edit, then click Copy (CTRL+C).
- 3) Put insertion point where you want to paste the information.
- 4) Click Edit, then click Paste (CTRL+V).

The information is copied from its original location and appears in its new location.

## Using bookmarks

You can set bookmarks to mark frequently accessed lines in your source file. Bookmarks are removed when the file containing them is closed or reloaded. Bookmarks store only the current line, not the column offset of the cursor. When a line containing a bookmark is deleted, the bookmark is also removed.

### To set a bookmark:

- 1) Move the insertion point to the line where you want to set a bookmark.
- 2) Right-click, then click Toggle Bookmark.  
An indicator appears in the margin next to the text.

### To set a bookmark at all lines that contain a specific string:

- 1) Right-click, then click Find.
- 2) In the Find What text box, enter the search string.
- 3) Click Mark All.  
An indicator appears in the margin of each line that contains the specified string.

### To remove a bookmark:

- 1) Move the insertion point to the line containing the bookmark you want to remove.
- 2) Right-click, then click Toggle Bookmark.  
The indicator disappears from the margin next to the text.

## ***How windows and regions work***

---

# ***Interacting with target processors***

---

The File menu commands let you: reset target processors, add files to a project, restart programs, and save projects.

*NOTE: If you load a session file on a target that is different from the type it was created on, CodeScape loads the session without loading the program file.*

*NOTE: You cannot add or remove targets during a session.*

### **Connecting to a target processor**

#### **Initialize a target**

When you run CodeScape it automatically connects to all detected targets (and prompts you to load the monitor if necessary).

#### **Reset a target**

You may be prompted to reset the target. If this occurs, do a soft reset. This restores the state of the target and re-initializes the monitors.

*NOTE: You may be prompted to reload the Program File after resetting the target.*

To do a Soft Reset:

- Click File, point to Reset, then click Soft Reset.  
-OR-
- In the Target window, right-click and point to Reset Target then click Soft Reset.

If a Soft Reset fails, you will be prompted to do a Hard Reset. This will reset the target and reload the monitors. You will be prompted to reload your project after a Hard Reset.

To do a hard reset:


- Click File, point to Reset, then click Hard Reset.  
-OR-
- In the Target window, point to Reset Target, then click Hard Reset.

*NOTE: You will be prompted to reload the monitor after a Hard Reset.*

## **Set the processor update rate**

Set the processor update rate to tell CodeScape when to update information to the target.

If the update rate interrupts the target causing jitter in your program:

- 1) On the Processor Combo toolbar click .  
The Processor Update Rate dialog box appears.
- 2) Select the Target that you want to set the update rate for.
- 3) Select the Processor on which your program is loaded.
- 4) Do one of the following:
  - Set the slider to Min.
  - OR-
  - Select Disable updates to this processor, to stop all region displays from updating.
- 5) Click OK.

***NOTE:** When you set this option it automatically overrides the region update rate set in the Region Configuration dialog box.*

### **Add files to a project**

#### **Load a program file**

- 1) Do one of the following:
  - Click File, then click Load Program File (CTRL+SHIFT+C).
  - OR-
  - In the Target window, right click, then click Load Program File...
- 2) Select the Target from the Target list box.
- 3) Select the Processor from the Processor list box.
- 4) In the Program File text box, enter your program file's path and file name.
- 5) In the Load Options text box, select one of the following radio buttons: Load Binary Only, Load Symbols/Debug only, or Load Both Binary and Symbols/Debug.
- 6) Select any of the following options you require:
  - Optimize loading of Adjacent Sections to load adjacent binary sections at the same time. Enabled by default.
  - Unlock the Program File (uses a copy) to copy the program file when it is loaded. Disabled by default.

The copied file is named cpy followed by a number, for example cpy2, and is placed in your default temporary directory. If your system configuration file does not specify a default temporary directory, the copied file is placed in the current working directory. If you build your program using an external build utility, you must reload the program file to view your changes in CodeScape.
  - Use the symbol information from the Hitachi link file to get the symbol table from the Hitachi map file.

The map file must have the extension \*.map and be in the same directory as your program \*.exe.
  - Enable Reset Options to specify the options you want to use for re-loading the program file on the target. The default is Reset Enabled, Soft Reset.
  - Select Enable Run Options (disabled by default). Select: Run, to run the specified program file when it is loaded, or select Run to and enter an address or expression to run to, when the program file is loaded.
- 7) Click OK.

**NOTE:** *If you select Enable Run Options and no debug information appears in a Source region, compile all source files for your project with debugging turned on.*



## **Saving and loading binary**

### ***Move large blocks of data in and out of memory***

Use Save binary and Load binary to move large blocks of data in and out of memory. This is useful for loading and saving bitmaps, or processor specific code to a selected area of memory.

- 1) Do one of the following:
  - Click File, then click Load binary.
  - OR-
  - Click File, then click Save binary.The Write Binary to Memory dialog box appears.
- 2) Enter the Source file name.
- 3) Specify the start address.
- 4) Do one of the following:
  - Select End, then in the text box below, specify the end address.
  - OR-
  - Select Length, then in the text box below, specify the length of the address.
- 5) Click OK.

**NOTE:** *You cannot write Binary to sensitive areas of memory such as invalid memory areas, read-only memory, and memory reserved for the monitors. If a sensitive area of memory is within a specified range, a message appears prompting you that the area of memory was skipped.*

## ***Interacting with target processors***

---

### **Load the binary part of a program file**

- 1) Do one of the following:
  - Click File, then click Load Program File... (CTRL+SHIFT+C).
  - OR-
  - In the Target window, right click and point to Load Program File...The Load Program File dialog box appears.
- 2) Select the Target from the *Target* list box.
- 3) Select the Processor from the *Processor* text box.
- 4) Enter the location of the program file in the *Program File* text box.
- 5) Click Load Binary Only.
- 6) Click OK.


### **Load the symbolic debugging information part of a program file**

- 1) Do one of the following:
  - Click File, then click Load Program File (CTRL+SHIFT+C).
  - OR-
  - In the Target window, right click and point to Load Program File...The Load Program File dialog box appears.
- 2) Select the Target from the *Target* list box.
- 3) Do one of the following:
  - Select the Processor from the Processor list box. Type the name and path of the program file in the *Program File* text box.
  - OR-
  - Click Browse and find the required file.
- 4) Click Load Symbolic Debugging information Only.
- 5) Click OK.

## Restarting a program

Restart loads the binary part of the current program file and resets the PC to the entry point (if known). If the program file's last modification time has changed, symbolic information is loaded.

### Load the binary part of the current program file

- Click Debug, point to Execution then click Restart (CTRL+SHIFT+R).  
-OR-
- In the Target window, right-click and point to Execution, then click Restart.  
-OR-
- On the Debug toolbar, click .
- Right-click in any region, click Execution, then click Restart.



# ***Working with sessions***

---

The commands for working with sessions are on the File menu.

CodeScape automatically saves the following debug information when you save a session:

- The configuration of Windows and Regions.
- The Project build information.
- The path for locating source files.
- The directory that contains CodeScape's source files.
- Which program files to load on each target processor.
- Any breakpoints that have been set.
- Any expanded functions in Watch and Local Watch regions.

*NOTE: When you next open the session CodeScape will automatically load this information.*

*NOTE: If you load a session file on a target that is different to the type it was created on, it loads without the program file.*

*NOTE: You cannot add or remove targets during a session.*

## ***Working with sessions***

---

### **File menu commands**

To open a new session:

- 1) Click File, Session New.  
The New dialog box appears.
- 2) Enter a file name using the extension. SSN.
- 3) Click OK.

To open an existing session:

- 1) Click File, Session Open...  
The Open dialog box appears.
- 2) Select a location in the *Look in* text box.
- 3) Select a file in the *File name* text box.
- 4) Click OK.

**NOTE:** *If you open a new session you must load a program file.*

To save a session:

- Click File, Session Save.

To save a session with a new name:

- 1) Click File, Session Save As...  
The Save As dialog box appears.
- 2) Enter a location the in the *Save In* text box.
- 3) Enter a new Session file name in the *File name* text box.
- 4) Click OK.

To close a session:

- Click File, Session Close.

**NOTE:** *You may be prompted to save the current session before CodeScape opens a new session.*

### Recently used files list

The File menu displays a list of recently used session files.

To open a recent session file:

- 1) Click File.
- 2) Click the Session that you want to open from the list.

*NOTE: You cannot hide this list or change the number of files displayed.*

To exit CodeScape:

- Click File, Exit.

When you exit CodeScape, it prompts you to save the following debug information:

- The configuration of Windows and Regions.
- The Project build information.
- The path for locating source files.
- The directory that contains CodeScape's source files.
- Which program files to load on each target processor.
- Any breakpoints that have been set.

*NOTE: The next time that you open the session CodeScape can load this information for you.*





# ***Working with projects***

---

Use the Project menu to set up, make, and build your project.

The commands on the Project menu let you set up the following:

- A project build environment.
- An editor.
- The path for locating source files.
- A directory for CodeScape's source files.

*NOTE: CodeScape uses a project file (for GNU.C) this is a makefile to link compiled source files when you build your project.*

### **Setting up a project build environment**

To configure a project, specify the project build utility that you want to use, then provide it with a command line, a filename, and an environment file.

To configure, make, and build a project:

- 1) Click Project, then click Setup Project.
- 2) Do one of the following:
  - Type the project's name and path location in the Project File text box. (For example, makefile.)
  - OR-
  - Select a recent project from the Project File list. (For example, makefile.)
  - OR-
  - Click Browse, then search for a project file. (For example, makefile.)
- 3) Enter the project build utility's name and path location in the Program Build text box. (For example, make, or SNASMSH2.)
- 4) Type any command line parameters that should be passed to the make command in the Command Line Modifiers text box. (For example, -f for GNU make.)
- 5) Enter the project environment file's name and path location in the Environment File text box. This may be the same directory as the program build file.
- 6) Click OK to accept the project build environment.
- 7) Make the project current and build it in one of the following ways:
  - Click Project, then click Make.
  - OR-
  - On the Project Build window, right-click, click Make.
  - OR-
  - Press CTRL+M.

The Input / Output window Build tab appears and automatically displays the specified build utility's output about the build. Any standard format errors and warnings are shown in the Input / Output window Build tab.

If a build error occurs, double-click an entry to invoke the editor and open the source file at the line containing the error or warning. To advance to the next error or warning press F4. Some external editors do not support this option and will open without displaying the line at which the error occurred.

## **The environment file**

To create an environment file:

- 1) Start an MS-DOS window and create the environment that you require to run the project build file that may use, for example, Hitachi C, GNU C, or SNASMSH2.
- 2) Create a file that contains the environment strings required by the project build file. For example, to create a file that contains the environment strings required to run the Hitachi tools, type: **set>hitachi.env**

## **Making and building a projects**

Make the project current and build it in one of the following ways:

- Click Project, then click Make.
- OR-
- On the Project Build window, right-click, click Make.
- OR-
- Press CTRL+M.

The Project Build window appears and automatically displays the specified build utility's output about the build. Any standard format errors and warnings are shown in the Project Build window. If a build error occurs, double-click an entry to invoke the editor and open the source file at the line containing the error or warning.

*NOTE: The Target and Project Build windows can be docked at the top and bottom of the main window, or left free floating.*

### **Setting up an editor**

The default editor is CodeScape's Edit region where you can edit existing files and create new ones, but you can configure CodeScape to use an external editor.

CodeScape supports the following external editors: Notepad, MS-DOS Editor, Multi-Edit for Windows, Multi-Edit for DOS, Codewright, Brief, and Vi for MS-DOS/UNIX. You can add and remove editors in this list.

When you select a default external editor, CodeScape displays the following:

- The manufacturer's default installation path location, followed by the command to invoke the editor in the Editor Path text box.
- The editor's command line parameter to go to a line. For example, if you select Multi Edit for Windows, %f /L%l appears in the Editor Arguments text box.

When you open a file in the editor, CodeScape replaces %f with the file's name, and %l with the first line to go to in the file. (Older versions of CodeScape use xxxxx instead of %l to represent the first line of a file.)

If a build error occurs when you make and build your project, CodeScape replaces %l with the line number containing the error or warning and displays it in the Input / Output window Build tab. You can scroll through the information as it is generated, or press F4 to move through any listed errors one at a time.

If the editor you select does not support this option, leave this field blank.

***NOTE:** To remove an editor: in Editor Name list select the editor that you want to remove, click Remove. To edit the string, without removing the editor from the list, press Delete.*

## Setting up an external editor

- 1) Click Project, then click Setup Editor.
- 2) Enter the name of the editor in the Editor Name list.
- 3) Enter the editor's path location, followed by command followed to invoke the editor in the Editor Path text box.
- 4) Enter %f, then the editor's command line parameter to go to a line, then %l in the Editor Arguments text box.

When you open a file in the editor, CodeScape replaces %f with the file's name, and %l with the first line to go to in the file. (Older versions of CodeScape use xxxxx instead of %l to represent the first line of a file.)

If a build error occurs when you make and build your project, CodeScape replaces %l with the line number containing the error or warning and displays it in the Input / Output window Build tab. If the editor you select does not support this option, leave this field blank.

- 5) Do one of the following:
  - If you selected a Windows based editor, select the Editor Is Windows Based check box.
  - OR-
  - If you selected an MS-DOS editor, clear the Editor Is Windows Based check box.
- 6) Click OK.

If you have set up a new editor, CodeScape automatically adds it to the Editor Name list when you click OK.

*NOTE: To remove an editor: in Editor Name list select the editor that you want to remove, click Remove. To edit the string, without removing the editor from the list, press Delete.*

*NOTE: Always save files edited in an external editor before using CodeScape's Make option to compile and build your project in CodeScape.*

### **Setting up the project commands**

#### **Set the path for locating source files**

In the Source File Search Path dialog you can: set, change, or remove a directory path name for CodeScape to look for your program's project files.

- 1) Click Project, then click Edit Source Path...  
The Source File Search Path dialog box appears.
- 2) Do one of the following:
  - Type in the Path of the Source files.
  - OR-
  - Click Browse and select the required directory.
- 3) Click Add.

#### **Remove a path from the Source File Search Path**

- 1) Click Project, then click Edit Source Path...  
The Source File Search Path dialog box appears.
- 2) Do one of the following:
  - Type in the path of the Source files.
  - OR-
  - Click Browse and select the required directory.
- 3) Click Remove.

### Set a directory for CodeScape's source files

Set or change relative path name of the default directory for your fileserver based operations in the Set fileserver directory dialog box. (This can be the same as the directory you set in Source File Search Paths.)

Do the following:

- 1) Click Project, then click Set FileServer Root Directory.
- 2) Enter the Path of the default directory that you wish to use for fileserver operations.
- 3) Click OK.

*NOTE: If the display update rate interrupts the target when it is loading information to the fileserver directory on your computer, click Project then select Enable Fileserver Optimization.*

*NOTE: Any configuration commands you set are saved in a session file when you exit including software breakpoints and watches, and program update rates.*





# Debugging

---

Debugging operations are:

- Tracing through your program code.
- Checking variables and structures.
- Adding and configuring breakpoints to control program execution.
- Simulating a target's processor operations to optimize tight assembler loops in your program.
- Profiling to examine the run-time behavior of program files written for Hitachi SH series target processors.

*NOTE: If you enable high level optimization when you build your project the compiler output can make source-level tracing confusing.*

*NOTE: Before you edit a program file from a UNIX target, convert it to a DOS readable format using a utility such as `to_dos` (use `to_unix` to return the file to a UNIX format).*

*NOTE: The toolbars provide access to the main debugging functions. Use the Toolbar Configuration check box to show or hide toolbars.*

### **Debugging modes**

You can debug your program in one of two modes:

- Operating system (OS mode). In OS mode program execution resumes back to the console BIOS after the Debug Stub is loaded. This is achieved with the minimum of disruption to the processor context prior to loading the Debug Stub. However, the DBR register is loaded with the default exception handler after resuming.
- Processor (CPU mode). In CPU mode the full debug stub loads, then the context is set up with default settings allowing a debugging session to start. CPU debugging does not pass program control back to the Console BIOS, control remains within the Debug Stub monitor.
  - The DBR register is loaded with the default Debug Stub exception handler.
  - The VBR register is loaded with the default Debug Stub exception handler.
  - The stack pointer is loaded at 0x0d000000.
  - The status register block bit is cleared to 0.

### **Selecting OS mode or CPU mode**

A software mode selection flag in the Debug Adapter (DA) determines the debugging mode. The flag is stored in EEPROM on the DA and provides power cycling of the DA. The default mode for either a new DA, or a re-flashed DA is OS mode.

You can change the debugging mode in one of two ways:

- When you run CodeScape, it detects new and re-flashed DAs and asks you if you want to change the debugging mode.
- If you want to change the debugging mode at any other time you must run DACHECK. For information using DACHECK refer to the Help supplied with the software.


## Running and stopping programs

The run options are:

- Run all processors simultaneously
- Stop all processors simultaneously
- Run a program
- Run a program until it executes a specified address
- Run a program to the cursor position
- Stop a program

### Run all processors simultaneously


Do one of the following:

- Click Debug, point to Execution then click Run All (CTRL+F9).  
-OR-
- Right-click, point to Execution then click Run All.  
-OR-
- On the Debug toolbar, click .

**NOTE:** *Program execution will stop at a breakpoint, or if an error occurs.*


### Stop all processors simultaneously

To stop all programs running simultaneously, do one of the following:

- Right-click in the Target window, point to Execution then, click Stop All.  
-OR-
- On the Debug toolbar, click .
- OR-
- Click Debug, point to Execution then click Stop All.

**NOTE:** *All processors will stop immediately.*


### Run a program

- 1) In the Target region, select the processor where your program is loaded.
- 2) Do one of the following:
  - Click Debug, then click Run (F9).
  - OR-
  - Right-click, point to Execution then click Run.
  - OR-
  - On the Debug toolbar, click .

*NOTE: Program execution will run until you stop it, or if an error occurs.*

*NOTE: When your program is running you can stop it by pressing F9.*

### Run a program until it executes a specified address


- 1) Do one of the following:
  - Click Debug, point to Execution then click Run to Address (SHIFT+F9).
  - OR-
  - Right-click, point to Execution then click Run to Address.
  - OR-
  - On the Debug toolbar, click .
- 2) Type an address in the *Expression* text box of the Run to Address/Instructions dialog box.
- 3) Click OK.

The address is the name of a function or, an expression that resolves to an address.

You can add breakpoints using Run to Address at any time during program execution, program execution stops when a breakpoint is encountered. Program execution will stop at the specified address, or at a breakpoint or if an error occurs.

## Run a program to the cursor position

In an active Source or Disassembly region, do one of the following:


- Click Debug, point to Execution then click Run to Cursor (ALT+F9).  
-OR-
- Right-click, point to Execution then click Run to Cursor.  
-OR-
- On the Debug toolbar, click .

You can add breakpoints using Run to Cursor at any time during program execution, program execution stops when a breakpoint is encountered. Program execution will stop at the cursor position, or at a breakpoint, or if an error occurs.

**NOTE:** *In the Call Stack region, use Run to Cursor to return to a specific function outside of the current one.*

## Stop a program

To stop a program running, in the Target region, select the processor on which your program is loaded, then do one of the following:

- Right-click in the Target window, point to Execution then right-click then click Stop.  
-OR-
- On the Debug toolbar, click .
- Click Debug, point to Execution then click Stop.  
-OR-
- Press F9.

**NOTE:** *The processor will stop immediately.*

### **Stepping into (tracing) code**

Trace functions are available either on the debug toolbar or on shortcut keys.

You can choose either source level tracing in an active Source region, or instruction level in an active Disassembly region. If you trace without a Source or Disassembly region open, tracing acts as if a Disassembly region is open.

*NOTE: You can trace a program at any time while debugging. All trace operations immediately stop program execution.*


*NOTE: All trace operations can be interrupted by breakpoints.*

The trace options are:

- Single step a line of source code
- Force step a line of source code at the disassembly level
- Step over a line of source code
- Step out of a line of source or disassembled code (return from function)
- Undo a step
- Enable Animated Step Run (animate trace)
- Step run into a line of source or disassembled code
- Step run out of a line of source or disassembled code
- Step Run Until
- Restart processor execution
- Stop a program

### Single step a line of source code

Do one of the following:

- Click Debug, then click Step (F7).  
-OR-
- On the Debug toolbar click .

In an active Disassembly region (or any non-source region), the target executes the instruction at the PC.

In an active Source region, the target executes the instruction at the PC. It stops when all low-level assembly instructions generated by the single source instruction have been executed. This includes:

- All instructions for a source macro instruction.
- Any C instructions that generate several assembler instructions.


Subsequent source lines (called by the current source line) may be in a different function or file, as determined by the execution flow.

**NOTE:** *Execution trace history is generated when single stepping.*

**NOTE:** *A trap instruction is treated as a subroutine (a BSR or JSR). Trap 32 is reserved by CodeScape and treated as a single instruction. Use Forced Step Into to step into the trap 32 routine. Stepping into trap 32 may cause the monitor to fail.*

### Force step a line of source code at the disassembly level

Do one of the following:

- Click Debug, then click Forced Step Into (SHIFT+F7).  
-OR-
- On the Debug toolbar click .

In a Disassembly region, Forced Step Into causes individual assembly instructions to be traced one at a time where this is normally not allowed, for example stepping into a trap 32.


In a Source region, the target executes the instruction at the PC with the current register values then stops at each individually generated assembler instruction. Each individual assembler instruction is traced using the disassembly-level Single Step instead of the source-level Single Step.

Step into mechanism, that is a Trap, Line-A, Line-F, or subroutine is entered and program execution halted inside. In the case of a single source instruction generating many assembly instructions you will need to press SHIFT+F7 several times on the source instruction before progressing to the next source instruction.

### Stepping over code

Execution trace history is generated when stepping over. This is useful when you need to undo a step operation. Step Over performs a single step if Step Over is not relevant in the current context.

To step over a line of source code:

- Click Debug, then click Step Over (F8).  
-OR-
- On the Debug toolbar click .

In disassembled code, the target executes the instruction at the PC then stops. A Trap, JSR or BSR is treated as a single instruction and program execution halted on the next instruction in memory when the routine is complete.

In source code, the target executes the instruction at the PC then stops when the source file reference has changed. When stepping over a function call, the entire function is executed. Execution is halted on the next source line.


**NOTE:** *You cannot step over conditional branches.*

**NOTE:** *Execution trace history is generated when stepping over.*




### Step out of a line of source or disassembled code

Use Step Out to run to and stop at the end of the current function call.

- Click Debug, then click Step Out.
- OR-
- Right-click in a region, click Execution, then click Step Out.
- OR-
- On the Debug toolbar, click .
- OR-
- Press CTRL+F8.

### Undo a step

- Click Debug, then click Unstep (CTRL+F7).
- OR-
- On the Debug toolbar click, .

CodeScape keeps a history of trace actions. Trace history is built-up and discarded automatically.

When you Unstep, only the current state of the processor and memory contents are untraced. You can Unstep:

- Instructions that are executed as a series of individual disassembly instructions.
- Traces in Source and Disassembly regions as long as there is a trace history left.

**NOTE:** *You cannot Unstep blocks of instructions.*

**NOTE:** *Where code is stepped over, all trace history to this point is lost.*

### Enable Animated Step Run (animate trace)


Select Enable Animated Step Run (default) to update all regions as each instruction executes.

- Click Debug, then click Enable Animated Step Run.

CodeScape will trace instructions and display updated information in the active window until a breakpoint occurs, or until another command is issued, for example start/stop.

### Step Run In


Use Step Run In to run to then stop at the start of each successively nested function calls.

- Click Debug, then click Step Run In (SHIFT+F7).  
-OR-
- Right-click in a region, click Execution, then click Step Run In.  
-OR-
- On the Debug toolbar click .

CodeScape will run to the start of the next function and then stop.


### Step Run Out

Use Step Run Out to run to and stop after each successively nested function call has completed.

- Click Debug, then click Step Run Out (SHIFT+F8).  
-OR-
- Right-click in a region, click Execution, then click Step Run Out.  
-OR-
- On the Debug toolbar click .

CodeScape will run to the end of the current function and then stop.

### Step Run Until...


- 1) Do one of the following:
  - Click Debug, then click Step Run Until...  
-OR-
  - On the Debug toolbar click, .
- 2) Enter an expression to run to in the Expression Evaluator.

CodeScape will trace instructions and display updated information in the active window until a breakpoint occurs, or until another command is issued, for example start/stop.

**NOTE:** *If the expression evaluates to a zero result, tracing continues.*


### Restart processor execution

Restart loads the binary part of the current program file and resets the PC to the entry point (if known). If the program file's last modification time has changed, symbolic information is also loaded.

- Right-click in the Target window, point to Execution then right-click then click Restart.  
-OR-
- On the Debug toolbar, click .
- OR-
- Right-click in any region, click Execution, then click Restart.  
-OR-
- Press CTRL+SHIFT+R.

### Stop a program running

To stop a program running, in the Target region, select the processor on which your program is loaded, then do one of the following:

- Right-click in the Target window, point to Execution then right-click then click Stop.  
-OR-
- On the Debug toolbar, click .
- OR-
- Click, point to Execution then click Stop.  
-OR-
- Press F9.

### Breakpoints

CodeScape has extensive software and hardware debugging features including breaking on data accesses within memory ranges and on external peripheral access. All breakpoint operations can be performed at any time through the Configure breakpoint(s) dialog box.

Software breakpoints cause exceptions during program execution if encountered in a memory area other than the one they were defined in. For example, CodeScape reports an unknown exception if 0x0C000000, 0x0D000000 is an image of 0x8C000000, 0x8D000000 and a software breakpoint is inserted at address 0x0C00B000, then occurs at 0x8C00B000.

To avoid this, mark mirrored memory images as shared memory in the configuration file dali.cfg. The example below defines three areas of shared memory as the same (Tag:B). The mirrored ASE-breaks are hidden and a breakpoint symbol is shown at their addresses.

```
[SEGA KATANA MasterSH4EVA_SharedMemory]
    SharedMemory = 0x0C000000, 0x0CFFFFFF, Tag:B
    SharedMemory = 0x8C000000, 0x8CFFFFFF, Tag:B
    SharedMemory = 0xAC000000, 0xACFFFFFF, Tag:B
```



**NOTE:** *If you add a breakpoint that uses a register or variable name as its address, the expression is only evaluated the first time it occurs during program execution.*

Breakpoint options are:

- Adding breakpoints
- Adding a breakpoint at the current cursor position
- Removing breakpoints
- Removing all breakpoints
- Enabling and disabling breakpoints
- Resetting breakpoints
- Configuring breakpoints
- The breakpoint expression format

## Adding breakpoints

You can add a breakpoint in Source, Disassembly, Memory, and Watch regions. You can add breakpoints at any time during program execution. Program execution stops when a breakpoint occurs. Add breakpoints using the menus, shortcut menus, or toolbars.

When a breakpoint is set and enabled in a Source or Disassembly region, the breakpoint set icon, , appears in the first column. When a breakpoint is disabled, the breakpoint disabled icon, , appears in the first column. When a watched variable is visible in the Watch region, the watched variable icon appears. In a Memory region the background color of the specified address changes.

A breakpoint is set with the following default behavior:


- Code breakpoint execution is halted once it has been triggered and no other action, such as logging, is performed. Code breakpoints are implemented in hardware if a ROM address is encountered or software otherwise.
- Watch breakpoints are triggered by any read or write data access to hardware. A message appears when the breakpoint has been triggered and all conditions have been met by default.

All breakpoint locations are tested to make sure that they are placed and configured correctly. If a problem is found a message appears prompting you to re-configure the breakpoint.

**NOTE:** *To change the default behavior of a breakpoint see To Configure a Breakpoint.*

**NOTE:** *You can add a breakpoint only to a line that generates code. (Shown by a '.' in column one of a Source region or Watch region, or at any point in the Disassembly region.)*

To add and run to a code breakpoint:


- 1) Right-click, click Goto Address.
- 2) On the Breakpoint toolbar, click  to set a breakpoint.
- 3) Right-click, click Execution, click Run. Your program will run until the breakpoint occurs.

**NOTE:** *You can set a maximum of two Hardware breakpoints for each SH2 processor on a Sega Saturn target.*

### Add a breakpoint at the current cursor position


In a Source or Disassembly region you can add code breakpoints. In a Watch or Memory region you can add data breakpoints.


In any region, place the cursor in the required position then:

- Click Debug, then point to Breakpoints then click Toggle Breakpoint (F5).  
-OR-
- Right-click, point to Breakpoints then click Toggle Breakpoint.  
-OR-
- On the Breakpoint toolbar, click .

### Removing breakpoints


In any region, place the cursor on the required breakpoint then:

- Click Debug, point to Breakpoints then click Toggle Breakpoint (F5).  
-OR-
- Right-click, point to Breakpoints then click Toggle Breakpoint.  
-OR-
- On the Breakpoint toolbar, click .
- -OR-
- In the Configure breakpoint(s) dialog box (CTRL+F5), select the breakpoint that you want to disable then click Remove.

The breakpoint set icon, , will disappear from the code window.

### Remove all breakpoints

In any region, place the cursor on the required breakpoint then:

- Click Debug, point to Breakpoints then click Remove all Breakpoints (SHIFT+F5).  
-OR-
- On the Breakpoint toolbar, click .
- -OR-
- Right-click, point to Breakpoints then click Remove all Breakpoints.  
-OR-
- In the Configure breakpoint(s) dialog box, click Remove All (CTRL+F5).

### Enable a disabled breakpoint


In any region, place the cursor on the required breakpoint then:

Click Debug, point to Breakpoints then click Enable Breakpoint.

-OR-

- Right-click, point to Breakpoints then click Enable Breakpoint.

-OR-

- On the Breakpoint toolbar, click .

-OR-

- In the Configure breakpoint(s) dialog box (CTRL+F5), click Code Settings and select Breakpoint Enabled.

The breakpoint set icon will change from  to  to show that the breakpoint is enabled.

### Disable an enabled breakpoint

In any region, place the cursor on the required breakpoint then:

- Click Debug, point to Breakpoints then click Disable Breakpoint.

-OR-

- Right-click, point to Breakpoints then click Disable Breakpoint.

-OR-

- On the Breakpoint toolbar, click .

-OR-

- In the Configure breakpoint(s) dialog box (CTRL+F5), click Code Settings and clear Breakpoint is Enabled.

The breakpoint set icon will change from  to  to show that the breakpoint is disabled.

### Enable all breakpoints


Do one of the following:

- Click Debug, point to Breakpoints then click Enable all Breakpoints (CTRL+SHIFT+F5).

-OR-


- Right-click, point to Breakpoints then click Enable all Breakpoints.

-OR-

- On the Breakpoint toolbar, click .


### Disable all breakpoints

Do one of the following:

- Click Debug, point to Breakpoints then click Disable all Breakpoints (CTRL+ALT+F5).  
-OR-
- Right-click, point to Breakpoints then click Disable all Breakpoints.  
-OR-
- On the Breakpoint toolbar, click .

### Reset all breakpoints

Do one of the following:

- Click Debug, point to Breakpoints then click Reset all Breakpoints (ALT+F5).  
-OR-
- Right-click, point to Breakpoints then click Reset all Breakpoints.  
-OR-
- On the Breakpoint toolbar, click .

*NOTE: Resetting all breakpoints sets all conditional values, including the current count, to their starting conditions.*

### Reset the trigger count for a breakpoint

- In the Configure breakpoint(s) dialog box select the breakpoint, click Reset.

### Reset only the current value of the count for a breakpoint


- In the Configure breakpoint(s) dialog box select the breakpoint, click General Conditions and click Reset Current.



## Configuring breakpoints

CodeScape enables breakpoint configuration including data accesses within memory ranges and breakpoints on external peripheral devices.

To configure a breakpoint:

- Click Debug, point to Breakpoints then click Configure Breakpoint(s)... (CTRL+F5).  
-OR-
- Right-click, point to Breakpoints then click Configure Breakpoint(s)...  
-OR-
- On the Breakpoint toolbar, click .

*NOTE: Software breakpoints cause exceptions during program execution if encountered in a memory area other than the one they were defined in.*

*NOTE: You can add a breakpoint and configure it manually using the Configure breakpoint(s) dialog box.*

*NOTE: Watch breakpoints trigger on data access, and code breakpoints trigger on the fetch-execute phase of the instruction cycle.*

### The Configure breakpoint(s) dialog box

In the Configure breakpoint(s) dialog box you can:

- Add, remove, and configure code and watch breakpoints.
- Enable or disable a breakpoint, set its location, and the resources it will use.
- Specify when a breakpoint will occur.
- Configure a prompt for when a breakpoint occurs.

### Using the Code Settings tab

Code breakpoints trigger on instruction execution. When a code breakpoint triggers the PC is at the same instruction in the pipeline. The Code Settings tab becomes available when you add or select a code breakpoint to configure.

- 1) Do one of the following:
  - Select a code breakpoint to configure from the list.
  - OR-
  - Click code to add a code breakpoint to configure.
- 2) Select Breakpoint Enabled (default), to enable a breakpoint.

You may be prompted to re-configure a disabled breakpoint. This can occur during code execution, restoring sessions, or when attributes could not be validated when configuring commands within this dialog box. A disabled breakpoint does not affect code execution or use any hardware resources.
- 3) Specify the position in memory where the code will stop on execution. In the *Location Expression* text box:
  - Enter the required expression.
  - OR-
  - Click Define. The Breakpoint Location Expression dialog box appears. Evaluate the expression to set the location address.
- 4) Then do one of the following:
  - Select C/C++, to use C/C++ expression syntax.
  - OR-
  - Select Assembly, to use SHx assembly language syntax.
- 5) In the Implementation mechanism group box:
  - Select Automatic and CodeScape will manage breakpoint resources. Breakpoints are implemented in software by default. If this is not possible then hardware resources are used.
  - OR-
  - Select Software to specify a software breakpoint.
  - OR-
  - Select Hardware to set a hardware breakpoint that is specific to your target processor.

**NOTE:** You can set a maximum of two Hardware breakpoints for each SH2 processor on a Sega Saturn target.

### ***Using the Watch Settings tab***

Watch (data) breakpoints trigger on memory data access. When a Watch breakpoint triggers the PC is several instructions ahead of that breakpoint in the pipeline.

The Watch Settings tab becomes available when you select or add a watch breakpoint to configure.

- 1) Do one of the following:
  - Select a watch breakpoint to configure from the list.
  - OR-
  - Click Watch to add a watch breakpoint to configure.
- 2) Select Breakpoint Enabled (default), to enable a breakpoint.

You may be prompted to re-configure a disabled breakpoint. This can occur during code execution, restoring sessions, or when attributes are not validated during command configuration in this dialog box. A disabled breakpoint does not affect code execution or use any hardware resources.
- 3) Specify the position in memory where the breakpoint is accessed. In the Location Expression text box:
  - Enter the required expression.
  - OR-
  - Click Define. The Breakpoint location expression dialog box appears. Evaluate the expression to set the location address.
- 4) Select Include Data Condition to change the Watch Access breakpoint into a Watch Data breakpoint that uses the features of the UBC (User Break Controller). Enter the required Data Expression, then click Define. The Breakpoint watch data expression dialog box appears. Evaluate the expression to set the location address.
- 5) Then do one of the following:
  - Select C/C++, to use C/C++ expression syntax.
  - OR-
  - Select Assembly, to use the Assembler's expression syntax.
- 6) In the Implementation mechanism group box:
  - Select Automatic and CodeScape will manage breakpoint resources. Breakpoints are implemented in software by default. If this is not possible then hardware resources are used.
  - OR-
  - Select Software to specify a software breakpoint.
  - OR-

## **Debugging**

---

- Select Hardware to set a hardware breakpoint that is specific to your target processor.
- 7) Under Access Size, enter the Access Size required (the default is Any). When you use Toggle to add a watch breakpoint its size, if known, will be used instead of Any.
  - 8) Under Access Type, select the Access Type required. The default is Both read and write access.

*NOTE: If you place a watch(data) breakpoint on a member of a union it will trigger for all members of that size, regardless of type. This also applies to anonymous unions, except that two members of the same size appear as two variables sharing the same address in memory.*

### **Using the General Conditions tab**

The General Conditions tab is for defining conditions that must be valid before a breakpoint is triggered. You can condition a breakpoint by memory access type and data value, and confirm that it executed on the correct trigger count.

**NOTE:** *To use a conditional expression select Include Conditional Expression.*

- 1) Do one of the following:
  - Enter a valid expression in the Include Conditional Expression text box.
  - OR-
  - Click Define to open the Breakpoint condition expression dialog box, then define the expression.
- 2) Do one of the following:
  - Select C/C++, to use C/C++ expression syntax.
  - OR-
  - Select Assembly, to use the Assembler's expression syntax.

The expression is evaluated for a logical result where a value of zero represents false and non-zero values represent true.
- 3) Select Include Trigger Count Condition to include the trigger condition. The condition is true when the Current Count reaches the specified Trigger Count value.
- 4) Enter the value for the Current Count to reach to make the Trigger Count Condition true.
- 5) Under Counters, check that the value in the Current box matches the value you set in the Trigger box. Click Reset Current to return the current count to zero.
- 6) Select when to increment the count. The default is to increment the Current Count whenever the breakpoint occurs or is evaluated.
- 7) If both expression and count conditions are included, select when to break in the expression. The default is OR.

### **Using the Trigger Actions tab**

Use the commands on the Trigger Actions tab to specify how CodeScape responds when a breakpoint has triggered.

Select any or all of the following radio buttons:

- Select Halt execution when conditions match to stop the program executing when the breakpoint conditions have been met. Clear this check box to continue execution after all other requested actions have been performed.
- Select Single shot - breakpoint is discarded when conditions match to discard the breakpoint after it has been triggered and all conditions have been met.
- Select Message box prompt when conditions match. CodeScape will display a message when the breakpoint has been triggered and all conditions have been met.
- Select Beep when conditions match. Your computer will beep when the breakpoint has been triggered and all conditions have been met.
- Select Cause processor simulation to and specify whether the Simulator should Start or Stop when the breakpoint has been triggered.
- Select Log Expression and choose either to produce a log when the breakpoint has been triggered or every time. Enter a valid Log expression.
- Run Script and specify a script to execute when the breakpoint conditions have been met.

*NOTE: If there is no Log region for the Target Processor, CodeScape creates one.*

*NOTE: You can only set one Start breakpoint and one Stop breakpoint for the Profiler.*

### **Using the Advanced tab to specify options for a code breakpoint**

*NOTE: The Location Address text box is read-only. To set the location, click the Code Settings tab.*

*NOTE: The ASID Mask Selector field is set to its default state and cannot be configured. It will be enabled in future releases.*

On the Advanced tab are commands for using the Hardware Implementation Mechanism. These commands apply only to Watch breakpoints and Code breakpoints.

- 1) Select Location Mask to specify which bits of the Location Address to mask out. Set Location Mask bits to 1 to ignore the corresponding Location Address bit, 0 otherwise.
- 2) In the *Break Mode* text box select either:
  - Before Execution.
  - OR-
  - After Execution.

### **Using the Advanced tab to specify options for a watch breakpoint**

*NOTE: The Location Address text box is read-only. To set the location, click the Code Settings tab.*

*NOTE: The ASID Mask Selector field is set to its default state and cannot be configured. It will be enabled in future releases.*

On the Advanced tab are commands for using the Hardware Implementation Mechanism. These commands apply only to Watch breakpoints and Code breakpoints.

- 1) Select Location Mask to specify which bits of the Location Address to mask out. Set Location Mask bits to 1 to ignore the corresponding Location Address bit, 0 otherwise.
- 2) In the *Data Mask* text box, set Data Mask bits to 1 to ignore the corresponding Data Address bit, 0 otherwise.
- 3) In the Bus cycle field, select the bus cycles to include, either CPU, or Peripheral (DMA), or both.

### ***Using the Global tab to specify the debug environment for Hitachi SH4-EVA processors***

On the Global tab are commands for setting the target processor's debug environment. The Global tab appears when you connect to an SH4-EVA target processor and you can specify any of the available options.

- 1) In the Global ASE Break Conditions for SH4-EVA CPU field:
  - Select Enable on-chip access detection and CodeScape will generate an on-chip I/O exception.  
The values displayed are the last on-chip address accessed, and the last on-chip data access when the exception occurred.
  - Select Enable break after LDTLB instruction execution and CodeScape will generate an LDTLB instruction break.  
The values displayed are the last PTEH loaded, and the last PTEL loaded into the MMU.
- 2) In the Global UBC Exception Handler Option field, select Use DBR vector (default). CodeScape will use the debug stub default exception handler for UBCs. This lets you define exception handling routines in your program, and to modify the VBR without affecting the behavior of UBC breakpoints.



## Breakpoint expression format

CodeScape has a powerful expression formatting facility for controlling the display of expressions in the Log tab on the Input / Output window.

Control formatting with expressions that work in a similar way to the C ``printf'` function. The expressions are numbered from 0 and can be any valid debugger expression referencing register names or memory locations. The syntax for a formatting expression is:

```
{ "FormattingString" | FormattingString } [ , C/C++Expression ]
```

Formatting string

A formatting string is a series of alpha numeric characters and three special format specifiers.

### Formatting string

A formatting string is a series of alpha numeric characters and three special format specifiers.

In the format `%[flags] [width] [.precision] type`, use the fields in the following ways:

Use the format:	To:
<code>\character</code>	Explicitly define a character. For example, <code>\$</code> displays a <code>\$</code> character.
<code>\$param_num</code>	Change the next argument index. For example, <code>\$0</code> sets the argument index to 0.
<code>%[flags] [width] [.precision] type</code>	Print a series of formatted characters and values to the Log tab on the Input / Output window. Type <code>%%</code> to print a single percent character.

- **[flags]** is an optional character or characters that control justification of output and printing of signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag can appear in a format specification.
- **[width]** is an optional number that specifies the minimum number of characters output.
- **[.precision]** is an optional number that specifies the maximum number of characters printed for all or part of the output field, or the minimum number of digits printed for integer values.
- **type** is a required character that determines whether the associated argument is interpreted as a character, a string, or a number.

### Flags specification

A flag directive is a character that justifies output and prints signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag directive may appear in a format specification.

Flag	Meaning
-	Left align the result within the given field width. The default is right align.
+	Prefix the output value with a sign (+ or -) if the output value is of a signed type. The sign appears only for negative signed values by default.
0	If width is prefixed with 0, zeros are added until the minimum width is reached. If 0 and - appear, the 0 is ignored. If 0 is specified with a none integer format (e.g. f, g, e) the 0 is ignored. The default is no padding.
blank ( ' ' )	Prefix the output value with a blank if the output value is signed and positive; the blank is ignored if both the blank and + flags appear. Default :No blank appears.
#	<p>When used with the o, x, or X format, the # flag prefixes any nonzero output value with 0, 0x, or 0X, respectively. Default: No blank appears.</p> <p>When used with the e, or f format, the # flag forces the output value to contain a decimal point in all cases. Default: Decimal point appears only if digits follow it.</p> <p>When used with the g or G format, the # flag forces the output value to contain a decimal point in all cases and prevents the truncation of trailing zeros. Default: Decimal point appears only if digits follow it. Trailing zeros are truncated.</p> <p>Ignored when used with c, d, i, u, or s.</p>

## Width specification

The second optional field of the format specification is the width specification. The width argument is a nonnegative decimal integer controlling the minimum number of characters printed. If the output value has fewer characters than the specified width, blanks are added to the right of the value unless the left align flag (-) is set. If width is prefixed with 0, zeros are added instead of blanks (not useful for left aligned numbers).

The width specification never causes a value to be truncated. If the number of characters in the output value is greater than the specified width, or if width is not given, all characters of the value are printed to the Log tab (subject to the precision specification).

If the width specification is an asterisk (\*), an int argument from the argument list supplies the value. The width argument must precede the value being formatted in the argument list. A nonexistent or small field width does not cause the truncation of a field; if the result of a conversion is wider than the field width, the field expands to contain the conversion result.

## Precision specification

The third optional field of the format specification is the precision specification. It specifies a nonnegative decimal integer, preceded by a period (.), which specifies the number of characters to be printed, the number of decimal places, or the number of significant digits. Unlike the width specification, the precision specification can cause either truncation of the output value or rounding of a floating-point value. If precision is specified as 0 and the value to be converted is 0, the result is no characters output, as shown below:

```
"%.0d", 0    /* No characters output */
```

If the precision specification is an asterisk (\*), an int argument from the argument list supplies the value. The precision argument must precede the value being formatted in the argument list.

### Type specification

Character	Type	Output Format
c	int	Single-byte character.
C	int	Single-byte character.
d	int	Signed decimal integer.
i	int	Signed decimal integer.
o	int	Unsigned octal integer.
u	int	Unsigned decimal integer.
x	int	Unsigned hexadecimal integer, using "abcdef."
X	int	Unsigned hexadecimal integer, using "ABCDEF."
e	double	Signed value with the form [-]d.dddd e [sign]ddd where d is a single decimal digit, dddd is one or more decimal digits, ddd is three decimal digits, and sign is + or -.
f	double	Signed value with the form [-]dddd.dddd, where dddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
g	double	Signed value printed in f or e format, whichever is more compact for the given value and precision. The e format is only used when the exponent of the value is less than -4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
G	double	Identical to the g format.
p	Pointer to	Prints the address pointed to by the argument in the form similar to %X (i.e. uppercase hexadecimal digits).
s	String	Specifies a single-byte-character string. Characters are printed up to the first null character or until the precision value is reached.
S	String	Specifies a single-byte-character string. Characters are printed up to the first null character or until the precision value is reached.

Character	Type	Output Format
i	Address	Displays (by disassembling) the op-code at the specified address.
l	Address	Displays (by disassembling) the op-code at the specified address with qualified symbol names if available.
t		Insert timestamp.
T		Insert timestamp.

## Examples

Expression	Description
"%X", pc	Evaluate and display the expression "pc" (this could be a variable or register) as an uppercase hexadecimal number. Output: 3b0
"0x%08x", \$pc	Evaluate and display the expression "\$pc" (this must be a register) as a lowercase hexadecimal number prepended by "0x" and padded with zeroes to 8 characters. Output: 0x000003b0
"0x%08x -> \$0 %l ", \$pc	Evaluate and display the expression "\$pc" (this must be a register) as a lowercase hexadecimal number prepended by "0x" and padded with zeroes to 8 characters followed by the disassembly of the op-code at that address with qualified symbol names. The \$0 reset the parameter index back to zero so the expression "\$pc" is used for both formatted options. Output: 0x000003b0 -> mov.l #BaseClass::i, r3

Evaluate and display the expression "\$pc" (this must be a register) as a lowercase hexadecimal number prepended by "0x" and padded with zeroes to 8 characters followed by the disassembly of the op-code at that address with qualified symbol names. The \$0 reset the parameter index back to zero so the expression "\$pc" is used for both formatted options.



# ***Simulating a target processor***

---

The Simulator is an optimizing tool for Hitachi SH series processors. It uses real targets for the Memory and Register regions.

When you single step in a Simulator region the cursor is shown at the instruction currently executing in the pipeline. During simulation the PC fetches instructions ahead of the current instruction. Some instructions are not executed because of changes in the program flow. For example, instructions fetched after a branch. When you single step in any other CodeScape region, the cursor is shown at the PC (program counter).

The Simulator enables you to optimize timing critical sections of Assembly code by simulating a target's processor operations. For example, you can set breakpoints to simulate a function that is part of a loop in your program.

**NOTE:** *You cannot run the Profiler and the Simulator at the same time.*

**NOTE:** *For details about processor pipeline operations, refer to the relevant Hitachi Programming Manual. For a copy of the manual, contact your Hitachi supplier, or connect to the Hitachi Japanese web site at <http://www.hitachi.co.jp>*

**NOTE:** *Memory timings do not model SDRAM banks (6000000-607FFFF, 6080000-60FFFFFF).*

## ***Simulating a target processor***

---

### **Using the Simulator's shortcut menu**

Select:	To:
Highlight Cache Misses	See in which slot a pipeline operation missed the cache.
Highlight Pipeline Stalls	See in which slot a pipeline operation stalled.
Show Stall Type	Show the type of stall generated.
Show Only Active Stages	Show active / all pipeline stages used.
Show Uppercase	Show instructions in upper case.
Show Symbols	Show operand values as symbols.
Show EAs & Lits.	Show the effective address and literals.
Source/Disassembly Tracking	Track the Simulator's cursor in source and disassembly regions.
Print	Print the results of program simulation.
Save to file...	Save the results of program simulation to a file.
Execution	Run, stop, and restart your program. Run your program to the cursor position, or until it executes a specified address. Run all of your program files simultaneously. Stop all of your programs running simultaneously. Use the single stepping options, or run the step options.
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.



## Running the Simulator

When you run the Simulator it generates information about the pipeline operation for each Assembly instruction. It also highlights any loss of performance in the processor cache and the pipeline. Use the Simulator's shortcut menu commands to configure the Simulator and access the debugging functions.

In the Target region, select the processor that you want to simulate:

- Select Debug, click Simulate Processor.
- OR-
- Right-click in the Target region, then click Simulate Processor.
- OR-
- Press CTRL+ALT+Z.

*NOTE: You cannot run the Profiler and the Simulator at the same time.*

## Running restrictions

The Simulator does not support the following features:

- DMA.
- Timers.
- Division unit.
- Power down mode.
- Memory mapped registers except for the CCR.
- External interrupts.

The Simulator disables external interrupts when it is running. If you use the sleep instruction you cannot wake the Simulator from sleep/standby mode.

Internal exceptions and interrupts are:

- Simulate NOP (no operation) and inform CodeScape of the appropriate exception.
- TRAPA 32 which is used for FileServer operation, software breakpoint operation, and hardware breakpoint operation.
- Address errors, illegal slot, and invoked instructions as reported on the processor status line.

## ***Simulating a target processor***

---

### **Debugging operations in the Simulator**

All of CodeScape's debugging functions are available when the Simulator is running. The debugging functions include commands for: controlling program execution, stepping code, using breakpoints, and setting the cursor to the PC and visa versa.

When you single step in a Simulator region the cursor is shown at the instruction currently executing in the pipeline. During simulation the PC fetches instructions ahead of the current instruction. Some instructions are not executed because of changes in the program flow. For example, instructions fetched after a branch. When you single step in any other CodeScape region, the cursor is shown at the PC (program counter).

### **Simulation results**

During program simulation the Simulator generates information about pipeline operation for each Assembly instruction. You can read any loss of processor performance from the simulation results shown in the Simulator's regions, or by printing the results.

## Information generated by the Simulator

When you simulate your project each instruction is executed in a Simulated slot (time). As the Simulator steps through time a linear description of pipeline operation is shown in its regions:

- The Address in memory for each line of source code.
- The Op-code for each instruction.
- The CPU time taken to execute each instruction at an address in memory.
- The Disassembly of the op-code for each instruction.
- Slot information for each stage of pipeline operation:
  - 1) The vertical cursor indicates the time taken by the processor to execute an instruction for each slot.
  - 2) The horizontal cursor indicates the instruction that is being allocated time in the active slot.
- Processor status information.

### Execution time

The execution time is the CPU time accumulated from the start of an instruction's 'ex' (execution) phase to the start of the next instruction's 'ex' phase.

### Processor status information

*Data displayed on the status bar for an active slot*

This status area:	Describes:
Diagnosis:	The type of stall encountered and what caused it.
Cache:	Cache memory operation stalls which occur when there is a read/write miss.
System clock:	The total time taken for processor operations upto the current cursor position.

**NOTE:** *The information generated by the Simulator can be saved in a configuration file with the extension \*.sim.*

## Simulating a target processor

---

### Pipeline interaction

The Simulator evaluates an instruction's functionality at the appropriate stage of the pipeline. The following instruction tells the processor to read 32 bits from the address stored in r0, then put the results in r3.

```
mov.L@r0,r3
```

When the instruction executes in a simulated slot (time) the following instruction stages are shown in the simulated pipeline:

```
IF | ID | EX | MA | WB
```

During the instruction's execution the following operations take place:

- At the IF stage the op-code for the instruction is read from memory.
- At the ID stage the instruction is decoded.
- At the EX stage instruction execution starts, and the contents of register r0 is read.
- At the MA stage memory is accessed at register r0 and the value is stored on the data bus.
- At the WB stage the value stored on the data bus is written back to memory at register r3.

#### *Instruction execution in the Pipeline region*

The mnemonic:	Indicates:
IF	Instruction fetch.
if	Dummy instruction fetch where external memory is not accessed.
ID	Instruction decoded / issued (All SH series processors.)Instruction decoded / issued / register read. (SH4 processors only.)
D	Decode stage locked.
d	Register read only. (SH4 processors only.)
EX	Instruction execution.
SX	Execution phase, the SX stage used.
SX*	SX stage locked not used.
NA	Memory not accessed / no operation address.
MA	Memory accessed / operation address.
MAm	Memory accessed / multiplier use. (SH2 processors only.)

The mnemonic:	Indicates:
mm	Multiplier busy. (SH2 processors only.)
WB	Register write back (data stored to registers after operation).
F0	Floating point 0 stage accessed. (Special Stage inner product / transforms).
F1	Floating point 1 stage accessed.
F1*	Floating point 1 stage locked and not accessed.
f1	Floating point 1 stage partial usage (can overlap with other f1's but not F1).
F2	Floating point 2 stage accessed.
F3	Floating point 3 stage accessed. (Special Stage divide / square root).
FS	Floating point store / writeback.
>FPSCR<	Floating point status register updated.

**NOTE:** In the \*.sim file all of the instructions are represented by the mnemonics listed above except, >FPSCR< which is represented by FC, and Mam which is represented Mm.

## ***Simulating a target processor***

---

### **Processor operation**

The CPU time accumulated by each slot is highlighted to show the state of the processor when an operation went off. Different colors and mnemonics' describe pipeline interaction at each phase.

*Processor operation in the Pipeline region*

<b>An operation colored:</b>	<b>Indicates that the processor:</b>
Black	Was OK
Red	Stalled
Blue	Missed the cache
Pink	Stalled and missed the cache

### **Pipeline stalls**

Where a stage from one instruction is in contention with a stage of the next or previous instruction a stall occurs. This slows down the operation of the pipeline. Simulation may show a stall in the processor's pipeline.

Remove a stall, in one of the following ways:

- Reorder the instruction sequence to remove an Instruction sequence stall.  
-OR-
- Move an instruction address in memory to remove an Instruction alignment stall.

**NOTE:** *For details about contention in instruction stages and execution states refer to the Hitachi Programming Manual for the 7600 Series.*

*Pipeline instruction stalls recognized by the Simulator*

<b>This symbol:</b>	<b>Shows this type of stall:</b>	<b>If possible, increase the speed of pipeline execution by:</b>
r>	A memory access conflicting with an instruction fetch. (SH1 and SH2 processors only.)	To align instructions that access memory on longword boundaries.
W>	A write back from the registry when a memory access is incomplete.	So that instructions that follow memory loads do not immediately use the same destination register.
x>	A multiplier usage stall. (SH1 / SH2 processors only.)	So that instructions that use the multiplier execute non-consecutively.
i>	An instruction generated stall.	You cannot do anything about this stall type. TRAP, TAS, RTE always stall.
R>	Two instructions trying to lock the same register. (SH4 processors only.)	So that instructions using the same register execute sequentially to ensure that they are not dual issued.
s>	The SX stage of the instruction being in use.	So that the instruction that locks the SX stage executes before instructions that use the SX stage non-consecutively.
f>	A floating point pipeline stall, caused by multiple use of the: F0, or F1, or F3 stages.	So that it uses instructions in the F0, or F1, or F3 stages once.
c>	One or more control group instructions being dual issued.	You cannot do anything about this stall type.
g>	Instructions of the same type occurring together and causing a dispatch failure.	So that instructions of the same type (such as EX + EX, LS + LS, BR + BR, FE + FE) do not occur together.
?>	An unknown stall type.	

**NOTE:** For details about pipeline instruction stalls refer to the *Hitachi Programming Manual for the 7600 Series*.

## ***Simulating a target processor***

---

### **Reading the results of simulation**

The Simulator generates information about pipeline operation for each Assembly instruction during program simulation. Any loss of processor performance appears in the results shown in the Simulator's regions. You can print the simulation results. (For an example of how to read the simulation results refer to the Simulator tutorial.)



# ***Profiling program files***

---

The Profiler is a powerful analysis tool that lets you examine the run-time behavior of program files written for Hitachi SH series processors.

You can configure the Profiler to analyze your program with two levels of detail: statistical and trace. The Profiler can help you to find out where your program spends its time, and how functions are called when it executes. You can use information generated by the Profiler to identify any inefficient sections of code.

*NOTE: To ensure accurate results, set the debug stub to run with the cache off when you trace profile. To do this, run DACHECK. For more information refer to the Help supplied with DACHECK.*


*NOTE: You cannot run the Profiler and the Simulator at the same time.*

### **Using the profiler: an overview**

#### **Open the Profiler and load a program file**

- 1) Click Tools, then click Profiler.  
The Profiler appears.
- 2) In the Target window, right-click, click Load Program File.  
The Load Program File dialog box appears.
- 3) Specify the Target, Processor, Program File, and Load Options.
- 4) Click OK.

#### **Profile the program file on the selected target processor**

- 1) 1 In the Profiler, right-click, click Setup...
  - Select Statistical for general profiling.
  - OR-
  - EVA for detailed information.
- 2) Click  to start profiling the selected program file.
- 3) In the Target window, right-click, select Execution, then click Run.

**NOTE:** *You can sort the profile data on the fly.*

**NOTE:** Click  to stop profiling the program file.

#### **Display a specific function(s)**

- 1) Double-click on the function that you want to profile to select (tag) it.
- 2) Right-click and select Trace Tree Profile Display to find out for the selected function: the functions that called it, and the functions it called.
- 3) Right-click, select Function Profile Filter, then click Show Tagged to switch the display from Show All functions to Show All Tagged functions  
The selected function will be the only function shown on the display.
- 4) Analyze the results.

**NOTE:** *You can tag more than one function to profile.*


### **View the source or disassembly of a specific function**

- 1) Double-click on the function that you want to view to tag it.
- 2) To view the:
  - Disassembly of the function, right-click then select Disassembly Display.
  - Source of the function, right-click then select Source Display.

### **Set and use a Profiler breakpoint**

- 1) Insert a Profiler Start breakpoint in one of the following ways:
  - Right-click in the Profiler Source Display, select breakpoints, then click Toggle Breakpoint.
  - OR-
  - Right-click in the Profiler Disassembly Display, select breakpoints, then click Toggle Breakpoint.

Profiling starts at the breakpoint insertion point. Profile information is generated only for the function that contains the breakpoint and the functions it calls.

- 2) In the Target window, right-click, click restart.
- 3) Click  to start profiling the selected program file.
- 4) In the Target window, right-click, select Execution, then click Run.

## Profiling program files

---











### The Profiler's commands

#### Using the Profiler's shortcut menu

Select:	To:
File	Load or save program profile information. Program profiles are saved using the extension *.prf.
Enable Profiler	Start or stop profiling your program file.
Enable Pass Between Breakpoints	
Remove All Profiler Breakpoints	Remove all Profiler breakpoints.
Trace Tree Profile Display	Find out for each function, the functions that called it, and the functions it called.
Function Profile Display	Find out for each function, the functions that called it, and the functions it called. Also, how much time your program spent in each function, and how many times each function is called.
Function Profile Filter	Arrange the view to show one of the following: all functions, all tagged functions, or all untagged functions.
Untag All	Untag all currently tagged functions.
Sort	Arrange the column view of the active Function Profile Display.
Source Display	View your program's original source code.
Disassembly Display	View your program at instruction level (assembly code).
Rename Function...	Enter a new name for a specific function.
Profiler Display Setup...	Specify the profile display options.
Setup...	Specify options for Statistical Profiling, or EVA Trace Profiling.

**NOTE:** To set a tag on a specific function, double-click it's entry in the Profiler.

**Options on the Profiler's toolbar**

To issue this command:	Click:
Start profiling the current program file.	
Stop profiling the current program file	
Toggle the display between a Trace Tree Profile and a Function Profile.	
Switch the display from Show All functions to Show All Tagged functions.	
Switch the display from Show All Tagged functions to Show All Not Tagged functions.	
Switch the display from Show All Not Tagged functions to Show All functions.	
Go to the next tagged function in the list.	
Display the program file's original source code.	
Display the program file at instruction level (assembly code).	
Toggle the sort options.	

**NOTE:** *If you want to Trace Profile, you must run the Profiler before you run your program.*

**NOTE:** *To set a tag on a specific function, double-click its entry in the Profiler.*

### **Debugging commands in the Profiler**

All of CodeScape's debugging functions are available when the Profiler is running. The debugging functions include commands for: controlling program execution, stepping code, using breakpoints, and setting the cursor to the PC and visa versa.

*NOTE: Profiler breakpoints are not standard breakpoints.*

### **Profiler breakpoints**

- Profiler breakpoints are not standard breakpoints. Note the following information about Profiler breakpoints:
- You can only set one Start Breakpoint, but you can set multiple Stop Breakpoints.
- Program execution does not stop when a Profiler breakpoint occurs.
- A frequently hit breakpoint increases the profile time.
- If you set a Profiler Start Breakpoint at the start of a function then the profiler only profiles that function and the functions it called. If an interrupt occurs during this function then it will be profiled.
- You do not have to set a Profiler Stop Breakpoint.

Insert a Profiler Start breakpoint in one of the following ways:

- 1) Right-click in the Profiler Source Display or the Profiler Disassembly Display.
  - 2) Select breakpoints, then click Toggle Breakpoint.
- OR-
- 1) Right-click in the Profiler Source Display or the Profiler Disassembly Display.
  - 2) Select breakpoints, then click Configure Breakpoint(s)...
  - 3) Select the Breakpoint that you want to configure.
  - 4) On the Trigger Actions tab select Cause processor profiling to, then:
    - Select Start to set a Profiler Start breakpoint.
- OR-
- Select Stop to set a Profiler Stop breakpoint.

### Tracing interrupt subroutines

When the Profiler is enabled program files execute more slowly than usual, but timer interrupts continue to trigger in real time. This can cause interrupts to trigger continuously inhibiting or even preventing Profiler from generating profile data outside of the exception handlers. If this occurs you may want to stop the Profiler profiling subroutines handled in interrupt and exception routines.

You can prevent interrupt subroutines from being traced and profiled in one of two ways:

- In the Profiler using the Interrupt Trace Subroutine Filter.  
Use this command when an interrupt or exception handler has few subroutine calls.  
-OR-
- In your program file using the disable and enable profiling BIOS calls commands.  
Use these commands when an interrupt or exception handler has many subroutine calls.

### Interrupt Trace Subroutine Filter

**CAUTION:** *Do not select the Interrupt Trace Subroutine Trace Filter check-box if disable and enable profiling BIOS calls are used in your program file.*

**NOTE:** *If an interrupt permanently triggers, use the disable and enable profiling BIOS calls commands in your interrupt routine.*

Use this command to stop the Profiler profiling subroutines handled in interrupt and exception routines.

### Disable and enable profiling BIOS calls

**CAUTION:** *Using these BIOS calls incorrectly causes invalid Profile data.*

**NOTE:** *Only use the disable and enable profiling BIOS calls commands with DA Firmware 4.4.0a onwards.*

The disable and enable profiling BIOS calls commands let you control subroutine profile tracing within your program file. To speed up the profile time, you can specify sections of your program for the Profiler to ignore.

## ***Profiling program files***

---

### **Profiler display types**

You can view the profile of your program by: function hits, function count (and children), function clock cycle (and children). Arrange the profile run-time order to view the functions in increasing order (Incremental), or decreasing order (Decremental).

### **Setting the display**

To specify the display options for the current profile:

- 1) Right-click in the Profiler, then click Profiler Display Setup.  
The Profiler Display Setup dialog box appears.
- 2) In the Column Data text box select one of the following options:
  - Display Both.
  - -OR-
  - Display Counts / Cycles.
  - -OR-
  - Display Percent.
- 3) In the Columns text box, select any of the following options:
  - Display Hits.
  - Display C1.
  - Display C1 + Children.
  - Display C2.
  - Display C2 + Children.
- 4) Click OK.

### **The Trace Tree Profile Display**

The Trace Tree Profile Display tells you for each function; the functions that called it, and the functions it called. A Trace Profile also shows, the total amount of time your program spends executing each function, and how much time it spends in each function and its children.

*NOTE: To ensure accurate results, set the debug stub to run with the cache off when you trace profile. To do this, run DACHECK. For more information refer to the Help supplied with DACHECK.*



## The Function Profile Display

The Function Profile Display lists the functions called. You can use the sort options on the shortcut menu to view the profile relative to: function hit, function count (and children), function clock cycle (and children). The Function Profile Filter lets you specify how you view tagged functions.

*NOTE: To tag a function, double-click its entry in the Profiler.*

## Searching for a function

To search for a function:

- 1) Move the insertion point to where you want to start searching from.
- 2) Type the Search string in the current profile.  
The Profiler automatically finds, and displays the nearest match.
- 3) To continue the search:
  - Press ENTER.
  - OR-
  - Press F3.

*NOTE: You can search for strings, whole words, or parts of words.*

*NOTE: The Profiler looks for exact matches first, then the nearest matches in descending order.*

*NOTE: The call tree automatically expands and contracts to display search results.*

## Changing the name of a specific function

To rename a function:

- 1) Select the function that you want to rename.
- 2) Right-click, click Rename Function...  
The Rename Function dialog box appears.
- 3) Enter the new name for the selected function.
- 4) Click OK.

## ***Profiling program files***

---

### **Profiling limitations**

*NOTE: The Profiler does not support JMP and BRA type instructions.*

The Profiler:

- Only receives Performance Counter information on the following events: JSR, RTS, RTE, BSR, BSRF, Interrupt, and Exception. It matches:
  - JSR, BSR, and BSRF with RTS.
  - Interrupts and exceptions with RTE.Any difference in the counter information is recorded and added to the totals.
- Ignores RTS unless it is preceeded and matched to a JSR.
- Cannot detect inline functions.
- Assumes RTE is a Context Switch (Task Swap) unless it matches the return address to a current task.
- Does not profile System Areas. The total time spent in the system areas is shown as:  
#TOTAL OF SYSTEM CALLS.
- Run slowly if your program file is complex and contains many functions.  
Large program files force the Profiler to use Virtual Memory. This increases the Profiler's memory access times from ~ 10 nano (E -09) seconds to ~ 10 mili ( E -03) seconds. This means the memory access times have dropped by an approximate factor of One Million.

To increase the speed of the Profiler, do one of the following:

- Use the Profiler breakpoints to specify a part of your program file to profile.
- Install more memory in your development computer.

# ***Viewing GD-M log information***

---

The Workshop tab on the Input/Output window displays GD-M log information, and lets you control the log events.

*NOTE: For more information, refer to How to emulate and test a GD project in the Help supplied with GDWorkshop.*

## Viewing GD-M log information

---

### Using the shortcut menu on the Workshop tab

Select:	To:
Disable Updates	Disable Workshop message logging.
Close Door	Close the door and start emulating a virtual CD.
SwitchToEmulator/Switch To GD-ROM	Toggle between the emulated GD-ROM image and the actual GD-ROM.
Nudge	Create a soft error on the next operation.
Hard Errors On	Enable hard errors as defined in Workshop. If you use this command, enable it before emulating.
Clear	Clear the log contents of the Workshop tab.
Allow Docking	Toggle docking for the window on or off.
Hide	Hide the window.

**NOTE:** For more information, refer to *How to emulate and test a GD project* in the *Help* supplied with *GDWorkshop*.

**NOTE:** If you enable hard errors during emulation you may not see errors in the log until after the next read command, because the data being read from the emulation may already be cached. If you enable hard errors before emulating you will see all the errors as they occur.

# ***Writing scripts to automate tasks***

---

CodeScape's script commands let you run Microsoft® JScript™ and VBScript macro scripts to automate routine tasks. CodeScape's script commands are demonstrated in example JScript and VBScript files. You can use the functions available in either script language to add commands of your own.

For details about using JScript and VBScript connect to the scripting area on the Microsoft Developer Network at: <http://msdn.microsoft.com/scripting>

- Using scripts
- Using the shortcut menu on the Script tab
- Adding a script to the menu
- Running a script
- CodeScape's scripting commands
- Expressing Numeric values and Numeric addresses
- Example VBScript
- Example JScript

# Writing scripts to automate tasks

---

## Using scripts

When you run a script the Input / Output window appears automatically and displays the Script tab with all messages generated by the current script.

To open the Input / Output window without running a script:

- Click View, Toolbar, then select the Input / Output check-box and click OK.

*NOTE: You can dock the Input/Output window at the top and bottom of the main window, or leave it free floating.*

## The shortcut menu on the Scripts tab

Select:	To:
Run Script	Select and run a script.
Clear	Clear the contents of the Script tab.
User Scripts	This option appears in gray until you add a script to the menu. When you add a script its name appears on the menu.
Allow Docking	Toggle docking for the window on or off.
Hide	Hide the window.

## **Adding a script to the menu**

When you add a script its name appears on the menu bar, and on the Script tab shortcut menu. You can add up to ten script files to run from either the menu bar, or the shortcut menu.

- 1) Click Tools, select Customize, then click Scripts...  
The Customize dialog box appears.
- 2) Click Add.
- 3) In the Menu Text box, enter the script name to display on menu.  
To remove an entry highlight the script's name in the Menu Text box and click Remove.
- 4) In the Menu Contents box, highlight the name of the script.
- 5) In the Script box, enter the path location and script file name.
- 6) Select either JScript, or VBScript to specify the script file type.
- 7) Do one of the following:
  - In the Arguments text box, enter any arguments to be passed to the script.  
Click OK.
  - OR-
  - Select the Prompt for arguments check-box.

*NOTE: Select a command in the Menu Contents box, then Use Move Up and Move Down to set where it appears on the Tools menu.*

*NOTE: To assign a keyboard shortcut to the script click Tools, select Customize then click Keyboard...*

## **Running a script**

- Click Tools, then select Scripts and click a script in the list.
- OR-
- On the Input / Output window, right-click on the Scripts tab, then click a script in the list.

*NOTE: Currently, scripts only support debugging a single target processor. When you run a script it automatically uses the selected target processor.*

*NOTE: A script that contains an infinite loop causes CodeScape to lock-up.*

### **Scripting commands**

#### **LoadProgramFile**

Loads the specified program file.

*Syntax*

**LoadProgramFile**(path and filename)

*Remarks*

This command uses the file path as a parameter and returns 1 if the file is loaded, else 0.

#### **HardReset**

Resets the target processor with a hard reset.

*Syntax*

**HardReset**( )

#### **SoftReset**

Resets the target processor with a soft reset.

*Syntax*

**SoftReset**( )

#### **Run**

Runs the target processor.

*Syntax*

**Run**( )

#### **WriteMessage**

Writes a message string to the script window.

*Syntax*

**WriteMessage**(string Message)



### WriteRegister

Sets the specified register to the given value.

*Syntax*

**WriteRegister**(Register value, Numeric value)

### RegisterValue ReadRegister

Gets the value held in the specified register.

*Syntax*

**RegisterValue ReadRegister**(RegisterName)

### LoadBinaryFile

Loads a binary file from the specified location.

*Syntax*

**LoadBinaryFile**(Path and filename, Numeric binary location)

### SetBreakpoint

Sets a code breakpoint at the specified address.

*Syntax*

**SetBreakpoint**(Numeric address)

### ClearAllBreakpoints

Clears all breakpoints.

*Syntax*

**ClearAllBreakpoints**( )

## Writing scripts to automate tasks

---

### RemoveBreakpoint

Removes the breakpoint from the specified address.

*Syntax*

**RemoveBreakpoint**(Numeric address)

### CreateBreakpoint

Creates a breakpoint of the given type at the address. Returns a breakpoint identifier on success, otherwise 0.

*Syntax*

**CreateBreakpoint**(Type,Address)

*Remarks*

Breakpoint type specifiers:

Breakpoint	Type
Code	0
Watch	1
Simulator or Start	2
Profiler start	3
Profiler stop	4

### EnableBreakpoint

Enables or disables the specified breakpoint.

*Syntax*

**EnableBreakpoint**(identifier,boolean enable)

*Remarks*

identifier: the breakpoint identifier.

enable: 1 to enable; 0 to disable.

## **SetBreakpointActions**

Enables or disables the specified breakpoint action.

### *Syntax*

**SetBreakpointActions**(identifier,numeric action,boolean enable)

### *Remarks*

enable: 1 to enable , 0 otherwise.

identifier: the breakpoint identifier.

Action	Value
Halt breakpoint when hit.	0
Remove breakpoint after being hit.	1
Display a message box prompt when hit.	2
Beep when hit.	3

## **SetBreakpointLog**

Sets a log expression for the breakpoint specified by the breakpoint identifier.

### *Syntax*

**SetBreakpointLog**(breakpoint identifier,string expression,boolean logType)

### *Remarks*

breakpoint identifier: the breakpoint identifier.

expression: the log expression.

logType: false to always log or true to log when conditions match.

### **SetBreakpointScript**

Attaches a script to a breakpoint.

#### *Syntax*

```
SetBreakpointScript(  
    identifier,  
    string script path,  
    numeric script type,  
    string script arguments,  
    boolean prompt)
```

#### *Remarks*

identifier: the breakpoint identifier.

script path: the file path for the script

script type: 0 for JScript and 1 for VBScript

script arguments: string holding the script's arguments

prompt: 1 to request arguments when the breakpoint triggers, 0 otherwise.

## **SetBreakpointCondition**

Sets a conditional expression for the breakpoint.

### *Syntax*

```
SetBreakpointCondition(  
    identifier,  
    string expression,  
    numeric expression type,  
    numeric trigger count,  
    boolean incOnTrue,  
    boolean breakWhen)
```

### *Remarks*

identifier: the breakpoint identifier.

expression: a string representing the condition.

expression type: 0 for C; non-zero for assembly

trigger count: the number of hits before breakpoint actions are performed.

incOnTrue: false to always increment the trigger count; true to increment the trigger count only when conditions are true.

breakWhen: false to break when the trigger reaches 0 or condition is true; true to break when trigger reaches zero and the condition is true.

**BOOL SetWatchBreakpointParameters**

Sets the parameters for a watch breakpoint.

*Syntax*

```
BOOL SetWatchBreakpointParameters(Identifier, Boolean  
incDataCondition, string dataCondition, numeric  
expressionType, numeric accessSize, numeric accessType)
```

*Remarks*

identifier: the breakpoint identifier.

incDataCondition: include a data condition.

dataCondition: expression specifying the data condition.

expressionType: the type of the specified expression.

accessSize: the access size. For example, byte, word, or long.

accessType: the type of access (read, write, or both).

Size	Value	Type	Value
Any	0	Read	1
Byte	1	Write	2
Word	2	Read or Write	3
Long	4		
Quad	8		

### **SetBreakpointLocationMask**

Select a location mask for the breakpoint.

*Syntax*

`SetBreakpointLocationMask(breakID,maskSelect)`

*Remarks*

Mask	Value
No bits masked	1
Lower 10 bits	2
Lower 12 bits	3
Lower 16 bits	4
Lower 20 bits	5
All bits	6

### **SetBreakpointDataMask**

Sets the data mask for a watch breakpoint.

*Syntax*

`SetBreakpointDataMask(identifier,mask)`

### **ReadByte**

Reads a byte from the specified area of memory.

*Syntax*

`ReadByte(Numeric address)`

## ***Writing scripts to automate tasks***

---

### **ReadWord**

Reads a word from the specified area of memory.

*Syntax*

**ReadWord**(Numeric address)

### **ReadLong**

Reads a long from the specified area of memory.

*Syntax*

**ReadLong**(Numeric address)

### **WriteByte**

Writes a byte from the specified area of memory.

*Syntax*

**WriteByte**(Numeric address, Numeric value)

### **WriteWord**

Writes a word from the specified area of memory.

*Syntax*

**WriteWord**(Numeric address, Numeric value)

### **WriteLong**

Writes a long from the specified area of memory.

*Syntax*

**WriteLong**(Numeric address, Numeric value)



**GetParam**

Returns a specific parameter.

*Syntax*

**GetParam**(short param)

**GetParamCount**

Returns the number of parameters passed to the script.

*Syntax*

**GetParamCount**( )

**IsRunning**

Returns 1 if running, 0 if not running.

*Syntax*

**IsRunning**( )

**ConfigureTraceHistory**

Specifies the events saved in the Trace history.

*Syntax*

**ConfigureTraceHistory**(numeric Setting,boolean Enable)

*Remarks*

Uses the settings:

Events	Setting
Log exceptions, interrupts, and rte	8
Log subroutines, bsr, bsrj, jsr, rts	4
Log branches, bf, bt, bf/s, bt/s, bra, braf,jmp	2

**DisplayTraceHistory**

Displays the current history in the script's window.

*Syntax*

**DisplayTraceHistory()**

*Remarks*

Uses the format:

Source	Destination		
0x0c010356	0x0c0103aa	rts	
0x0c0101e6	0x0c010350	rts	
0x0c0100e6	0x0c010128	bra	\$0c010128
0x0c01034c	0x0c010028	bsr	BigTest
0x0c0103a6	0x0c010334	bsr	struct_test
0x0c010280	0x0c0103a0	rts	
0x0c01039c	0x0c010214	bsr	BitFieldTest

**ClearDisplay**

Clear the script output window.

*Syntax*

**ClearDisplay()**

**Example VBScript**

```
' This script does not do anything useful other than demonstrate the
functions available

ClearDisplay
HardReset
SoftReset
DisplayParameters
LoadSomeBinary
LoadProgramFile( "d:\\projects\\maketest\\hello.elf" )
SetBreakpoint( "add_fn" )
ConfigureTraceHistory TH_LOGEXCEPT + TH_LOGSUB, true
Dim Running
Running = 1
Do
    Running=IsRunning
Loop Until Running = 0
DisplayTraceHistory
ReadSomeRegisters
WriteSomeRegisters
ReadSomeMemory
WriteSomeMemory
ReadSomeMemory
ClearAllBreakpoints
CreateCodeBP
ClearAllBreakpoints
CreateWatchBP
WriteMessage( "Script complete. Removing all breakpoints." )
ClearAllBreakpoints

'
' Breakpoint types
'
BPTYPE_CODE= 0
BPTYPE_WATCH=1
BPTYPE_SIMSTART=2
BPTYPE_PROFSTART=3
BPTYPE_PROFSTOP=4

'
' Breakpoint Actions
'
BPACTION_HALT=0
BPACTION_ONESHOT=1
BPACTION_PROMPT=2
BPACTION_BEEP=3

'
' Breakpoint Script Types
'
BPSCRIPT_JSCRIPT=0
BPSCRIPT_VBSCRIPT=1
'
```

## Writing scripts to automate tasks

---

```
, 'Breakpoint expression types
,
BPEXPR_C          = 0
BPEXPR_ASSEMBLY=1
,
, 'Breakpoint address masks
,

BPLOCMASK_NONE=1
BPLOCMASK_LOW10=2
BPLOCMASK_LOW12=3
BPLOCMASK_LOW16=4
BPLOCMASK_LOW20=5
BPLOCMASK_ALL=6
,
, 'Breakpoint access sizes
,

BPACCESSSIZE_ANY=0
BPACCESSSIZE_BYTE=1
BPACCESSSIZE_WORD=2
BPACCESSSIZE_LONG=4
BPACCESSSIZE_QUAD=8
,
, 'Breakpoint access types
,

BPACCESSTYPE_READ=1
BPACCESSTYPE_WRITE=2
BPACCESSTYPE_RW=3
,
, 'Trace history configuration options
,

TH_LOGEXCEPT=8
TH_LOGSUB=4
TH_LOGBRANCH=2
,
, 'Create a breakpoint on the 1K aligned block of memory that
, 'the symbol main resides in.
,
Sub CreateCodeBP()
    Dim breakID
    breakID=CreateBreakpoint( BPTYPE_CODE, "main" )
    SetBreakpointAction breakID, BPACTION_HALT, true
    SetBreakpointAction breakID, BPACTION_ONESHOT, false
    SetBreakpointAction breakID, BPACTION_PROMPT, false
    SetBreakpointAction breakID, BPACTION_BEEP, true
    SetBreakpointScript breakID,
    "e:\\projects\\codescape\\debugs\\testscript.js",
    BPSCRIPT_JSCRIPT, "arg1 arg2 arg3", false
```

```
SetBreakpointLog breakID, "Hello John", BPEXPR_C
SetBreakpointLocationMask breakID, BPLOCMASK_LOW10
SetBreakpointCondition breakID, "index == 375", BPEXPR_C, 37,
true, true
End Sub

Sub CreateWatchBP()
    breakID= CreateBreakpoint( BPTYPE_WATCH, "main" )
    SetWatchBreakpointParameters breakID, true,"14", BPEXPR_C,
BPACCESSSIZE_BYTE, BPACCESSTYPE_WRITE
End Sub

Sub WriteSomeRegisters()
    WriteRegister "fr0", 3.14159
    WriteRegister "r0", "0xabcdef"
    WriteRegister "pc", "main + 0x30"
End Sub

Sub ReadSomeRegisters()
    WriteMessage( "Value of pc = " & ReadRegister( "pc" ) )
    WriteMessage( "Value of r0 = " & ReadRegister( "r0" ) )
End Sub

Sub LoadSomeBinary()
    LoadBinaryFile "d:\\projects\\codescape\\satmon.bin",
"201392128"
    LoadBinaryFile "d:\\projects\\codescape\\satmon.bin", 201392128
    LoadBinaryFile "d:\\projects\\codescape\\satmon.bin",
"0xc010000"
    LoadBinaryFile "d:\\projects\\codescape\\satmon.bin", "main"
End Sub

Sub DisplayParameters()
    NumParams=GetParamCount
    WriteMessage( "Number of parameters = " & NumParams )
    For i = 1 To NumParams
        WriteMessage( "Parameter " & i & " = " & GetParam( i - 1 ) )
    Next
End Sub

Sub ReadSomeMemory()
    WriteMessage( "Byte at main = " & ReadByte( "main" ) )
    WriteMessage( "Word at main + 4 = " & ReadWord( "main + 4" ) )
    WriteMessage( "Long at main + 8 = " & ReadLong( "main + 8" ) )
End Sub

Sub WriteSomeMemory()
    WriteByte "main", 255
    WriteWord "main + 4", "0xabcd"
    WriteLong "main + 8", "0xfedcba"
End Sub
```

### **Example JScript**

```
// Note: this script does not do anything useful. It just
demonstrates the current
// script commands and how they can be called.
//
// Breakpoint types
//
BPTYPE_CODE= 0;
BPTYPE_WATCH=1;
BPTYPE_SIMSTART=2;
BPTYPE_PROFSTART=3;
BPTYPE_PROFSTOP=4;

//
// Breakpoint Actions
//
BPACTION_HALT=0;
BPACTION_ONESHOT=1;
BPACTION_PROMPT=2;
BPACTION_BEEP=3;

//
// Breakpoint Script Types
//
BPSCRIPT_JSCRIPT=0;
BPSCRIPT_VBSCRIPT=1;

//
// Breakpoint expression types
//
BPEXPR_C      = 0;
BPEXPR_ASSEMBLY=1;

//
// Breakpoint address masks
//

BPLOCMASK_NONE=1;
BPLOCMASK_LOW10=2;
BPLOCMASK_LOW12=3;
BPLOCMASK_LOW16=4;
BPLOCMASK_LOW20=5;
BPLOCMASK_ALL=6;

//
// Breakpoint access sizes
//

BPACCESSSIZE_ANY=0;
BPACCESSSIZE_BYTE=1;
BPACCESSSIZE_WORD=2;
BPACCESSSIZE_LONG=4;
BPACCESSSIZE_QUAD=8;

//
```

```
// Breakpoint access types
//

BPACCESSSTYPE_READ=1;
BPACCESSSTYPE_WRITE=2;
BPACCESSSTYPE_RW=3;

//
// Trace history configuration options
//

TH_LOGEXCEPT=8;
TH_LOGSUB=4;
TH_LOGBRANCH=2;

//
// Create breakpoint on the 1k aligned block of memory that the
symbol main resides in
//
function CreateCodeBP()
{
    breakID=CreateBreakpoint( BPTYPE_CODE, "main" );
    SetBreakpointAction( breakID, BPACTION_HALT, true );
    SetBreakpointAction( breakID, BPACTION_ONESHOT, false );
    SetBreakpointAction( breakID, BPACTION_PROMPT, false );
    SetBreakpointAction( breakID, BPACTION_BEEP, true );
    SetBreakpointScript( breakID,
    "e:\\projects\\codescape\\debugs\\testscript.js", BPSCRIPT_JSCRIPT,
    "arg1 arg2 arg3", false );
    SetBreakpointLog( breakID, "Hello John", BPEXPR_C );
    SetBreakpointLocationMask( breakID, BPLOCMASK_LOW10 );
    setBreakpointCondition( breakID, "index == 375", BPEXPR_C, 37,
    true, true );
}

function CreateWatchBP()
{
    breakID= CreateBreakpoint( BPTYPE_WATCH, "main" );
    SetWatchBreakpointParameters( breakID, true, "14", BPEXPR_C,
    BPACCESSSIZE_BYTE, BPACCESSSTYPE_WRITE );
}

function WriteSomeRegisters()
{
    WriteRegister( "fr0", 3.14159 );
    WriteRegister( "r0", "0xabcdef" );
    WriteRegister( "pc", "main + 0x30" );
}

function ReadSomeRegisters()
{
    WriteMessage( "Value of pc = " + ReadRegister( "pc" ) )
```

## Writing scripts to automate tasks

---

```
        WriteMessage( "Value of r0 = " + ReadRegister( "r0" ) )
    }

function LoadSomeBinary()
{
    LoadBinaryFile( "d:\\projects\\codescape\\satmon.bin",
"201392128" );
    LoadBinaryFile( "d:\\projects\\codescape\\satmon.bin",
201392128 );
    LoadBinaryFile( "d:\\projects\\codescape\\satmon.bin",
"0xc010000" );
    LoadBinaryFile( "d:\\projects\\codescape\\satmon.bin", "main" );
}

function DisplayParameters()
{
    NumParams=GetParamCount()
    WriteMessage( "Number of parameters = " + NumParams );
    for( i = 0; i < NumParams; i++ )
    {
        WriteMessage( "Parameter " + i + " = " + GetParam( i ) )
    }
}

function ReadSomeMemory()
{
    WriteMessage( "Byte at main = " + ReadByte( "main" ) );
    WriteMessage( "Word at main + 4 = " + ReadWord( "main + 4" ) );
    WriteMessage( "Long at main + 8 = " + ReadLong( "main + 8" ) );
}

function WriteSomeMemory()
{
    WriteByte( "main", 255 );
    WriteWord( "main + 4", "0xabcd" );
    WriteLong( "main + 8", "0xfedcba" );
}

ClearDisplay();
HardReset();
SoftReset();
DisplayParameters();
LoadSomeBinary();
LoadProgramFile( "d:\\projects\\maketest\\hello.elf" );
SetBreakpoint( "add_fn" );
ConfigureTraceHistory( TH_LOGEXCEPT + TH_LOGSUB, true );
Run();
while( IsRunning() != 0 )
{
    ;
}
DisplayTraceHistory();
ReadSomeRegisters();
WriteSomeRegisters();
ReadSomeMemory();
WriteSomeMemory();
```



```
ReadSomeMemory();  
ClearAllBreakpoints();  
CreateCodeBP();  
ClearAllBreakpoints();  
CreateWatchBP();  
WriteMessage( "Script complete. Removing all breakpoints." );  
ClearAllBreakpoints();
```



# ***Evaluating expressions***

---

The expression evaluator dialog is used for several operations, including: Edit Register, and Goto Address. In the dialog you can use the C/C++ expression evaluator or the Assembler's expression evaluator.

## Evaluating expressions

---

### Expression evaluator dialog box (ALT+E)

The expression evaluator is a general purpose dialog used for several operations, including: edit register, edit memory value, edit local value, edit watch value, and goto address.

#### *The options on the Expression Evaluator*

Use the:	To:
Expression Combo box	Edit an existing expression, or select one from the history list.
Result field	View the results of an expression evaluation including any error messages.
ExpressionFormatradio buttons	Select C/C++, or Assembly as the expression format.
Default radix radio buttons	Select binary, octal, decimal, or hex, as the radix to use for the expression, or specify another radix in the Other text box. For C expressions this permits only control of the output radix.
Evaluate button	Evaluate an expression in the Expression Combo text box.
Symbol button	Use the Symbol Completion dialog box to search for a symbol from those available in the program file.
File button	View a list of all the files used to build the program file in the List Files in Program File dialog box. The dialog box also provides access to the address for "file:line number" information.
Lock check box	Lock the current expression to a file or symbol.

**NOTE:** When you evaluate assembler expressions in a watch region the maximum number of characters you can enter is 127.

## Symbol Completion dialog box (ALT+S)

Use the Symbol Completion dialog box to search for a symbol in the program file.

### *Options on the Symbol Completion dialog box*

Use the:	To:
Find String text box	Enter the first few characters of the symbol to search for.
Only Search For Symbols Within Scope check box	Search for symbols in scope (select the check box), or to search for symbols in the whole program (deselect the check box).
Include Linkage Level Symbols check box	Include low level symbols in the search (select the check box). The default is deselected.
Possible Completions text box	View a list of all symbols that match the current Search String.
Lookup button	Click Lookup to start another search.
OK button	Accept the current search string.
Cancel button	Ignore current search string.

### C/C++ expressions

The C/C++ expression evaluator accepts expressions in a C-like format.

#### *Operator precedence*

Operator	Type	Usage	Description
( )	Primary		Parenthesis Brackets
[ ]	Primary	pointer[expr]	Subscripting
.	Binary	object.member	Member selection
->	Binary	pointer->member	Member selection
sizeof()	Unary	sizeof(expr)	Size of object.
sizeof()	Unary	sizeof(type)	Size of type
-	Unary	- expr	Unary Minus
+	Unary	+ expr	Unary Plus
~	Unary	~ expr	Bitwise NOT
!	Unary	! expr	Logical NOT
*	Unary	* expr	De-reference
&	Unary	& lvalue	Address of
*	Binary	expr *expr	Multiply
/	Binary	expr/expr	Divide
%	Binary	expr % expr	Modulo (remainder)
+	Binary	expr + expr	Add (plus)
-	Binary	expr - expr	Subtract (minus)
<<	Binary	expr << expr	Shift Left
>>	Binary	expr >> expr	Shift Right
<	Binary	expr < expr	Less than

Operator	Type	Usage	Description
<=	Binary	expr <= expr	Less than or equal
>	Binary	expr > expr	Greater than
>=	Binary	expr >= expr	Greater than or equal
==	Binary	expr == expr	Equal
!=	Binary	expr != expr	Not Equal
&	Binary	expr & expr	Bitwise AND
^	Binary	expr ^ expr	Bitwise Exclusive OR
	Binary	expr   expr	Bitwise Inclusive OR
&&	Binary	expr && expr	Logical AND
	Binary	expr    expr	Logical Inclusive OR

*Operands that the C/C++ operators act on*

Operand	Definition
Constants (Floating or Integer)	Constants can be: hexadecimal numbers prefixed with '0x'. Octal numbers prefixed with '0', or unsigned numbers postfixed with a 'U'. Characters, for example 'A', are not accepted.
Registers	The name of a valid register.
Symbols	Symbol names take into account their type. For example a variable defined as (char chr = 'A') would return 'A' when evaluated. To get the address of the object '&chr' is required.

## ***Evaluating expressions***

---

### **Operator limitations:**

- Typecasts. Typecasts of basic type, such as `int`, `float`, `unsigned int`, `int *`, `char *`, are valid. Typecasts to user defined type such as, `struct basic *`, are not valid.
- Scope operator, `::`. The scope operator is valid as part of a class element name, for example, `c_basic::print`.
- Assignment operators, such as `=`, `+=`, `*=`, `++`, `--`, are not implemented in this release.
- File/line number format is not implemented in this release.



## Assembler expressions

The assembler expression evaluator is fully compatible with SNASM2.

*Operator precedence:*

Operator	Type	Usage	Description
( )	Primary	(expr)	Parenthesis Brackets
[ ]	Primary	[expr]	Address of
-	Unary	- expr	Negative expr
+	Unary	+ expr	Positive expr
~	Unary	~ expr	Bitwise NOT
<<	Binary	expr << expr	Shift left
>>	Binary	expr >> expr	Shift right
&	Binary	expr & expr	Logical AND
!	Unary	! expr	Logical NOT
	Binary	expr	Logical Inclusive OR
^	Binary	^ expr	Logical Exclusive OR
*	Binary	expr * expr	Multiply
/	Binary	expr / expr	Divide
%	Binary	expr % expr	Modulo (remainder)
+	Binary	expr + expr	Add (plus)
-	Binary	expr - expr	Subtract (minus)
=	Binary	expr = expr	Equals
<>	Binary	expr <> expr	Not Equals
<	Binary	expr < expr	Less Than

## Evaluating expressions

---

Operator	Type	Usage	Description
<=	Binary	expr <= expr	Less Than or Equals
>	Binary	expr > expr	Greater Than
>=	Binary	expr >= expr	Greater Than or Equals

*Operands that the assembly operators act on*

Operand	Definition		
Constants (Integer)	Constants can be defined in several operators to denote different radix:		
	Variable Hex Decimal Binary	X_<number>	where X is a single digit base prefix '\$' or '0x' postfix 'h' prefix '#' postfix 'd' prefix '%' postfix 'b'
Registers	The name of a valid register.		
Symbols	Symbols are evaluated to labels, so a variable of type(char chr = 'A'), would return the address of (label to) the variable A when evaluated. Labels can be qualified by: 'b', 'w', 'l' for byte, word or long respectively [symbol]@b, [symbol]@w, [symbol]@l :<number> for the filename line number		

# ***Using the command-line***

---

Use the command-line commands to specify how CodeScape will run. For example, CodeScape can run from another application such as the Codewright editor, or from a batch file.

To run CodeScape from the command-line, type CodeScape then one or more optional switches. Always separate switches a space, but do not use spaces within the argument of a switch.

The syntax is:

```
codescape[Switch]...
```

## **Files used by CodeScape**

Filename	Description
Session	This file contains the information needed to restore a previous debugging session.
Program	The object file. This contains binary and optionally, source level debug and symbol table information produced by the assembler or compiler.

---

To change file or folder properties:

- 1) Click the file or folder whose properties you want to change.
- 2) On the File menu, click Properties.

**NOTE:** You can drag a file's icon into a document, or even drag a shortcut icon.

## Command-line switches

Use this switch:	To:
<code>[-l/?]</code>	Run CodeScape and view Help on using the command-line.
<code>[-l/]nologo</code>	Run CodeScape without the splash screen appearing.
<code>[-l/]c</code>	View information on the Log tab as each command of the Cross Products Fileserver library (libcross) executes. (Crosslib Verbose mode.)
<code>[-l/]i=Session</code>	Run CodeScape and open the session file "Session". The session file specifies: target connections, object files used by each target, processor update rates, breakpoints, watch expressions, log expressions, and window positions and displays. If memory ranges are not set, CodeScape looks for them in DEFAULT.SSN. (Use Project Info.)
<code>[-l/]autoload</code>	Run CodeScape using the last loaded session file.
<code>[-l/]noautoload</code>	Run CodeScape without opening the last loaded session file. Use this option to override autoload specified in a batch file.
<code>-nomake</code>	Disable the project make facility. If CodeScape is running, it ignores this option.
<code>-s=script[,param-list]</code>	Run a script with the given (optional) parameter list
<code>-nogui</code>	If none of the other options specified require the gui to be present, exit CodeScape after loading and starting the program(s). If CodeScape is running, it ignores this option.

## Loading a program file from the command-line

When you load a program file from the command-line, use the switch format:

```
-t#p#[b][n][ r+ | r- | r(expression) ][ h | s ][c+ | c- ][ l+ | l- ]:Program
```

You must specify the target (t#) and processor (p#) to use, and the program file to load. The processor is identified by its processor ID # (0-7) where 1=Master and 2=Slave. The program file is specified by "program".

Optional switches for loading a program file

Use this switch:	To:
b	Download the binary from the object file.
n	Suppress debug information. The n option does not require symbols. If you use the n option without the b option it has no effect.
r+	Load and run the program file.
r-	Load, but don't run the program file.
r(expression)	Load and run the program file, then break at the address specified. "expression" is usually a symbol such as "main".
h	Perform a hard reset of the target, then load the program file.
s	Perform a soft reset of the target, then load the program file.
c+	Concatenate the sections in the program file.
c-	Don't concatenate the sections in the program file.
l+	Lock the program file.
l-	Don't lock the program file.

The following options are mutually exclusive:

- The run options: r+, r-, and r(expression).
- The reset options, h and s.
- The load options, c+ and c-.



# ***Appendix A: Frequent operations***

---

Keyboard shortcuts are available for frequently used debugging operations. All operations are supported by Access keys which are shown on each menu item by an underlined letter.

To use the keyboard to access CodeScape's commands:

- Press F10, select an item with the cursor keys, press ENTER.
- OR-
- Press the menu's keyboard shortcut, select an item with the cursor keys, press ENTER.

## Appendix A: Frequent operations

---

### File menu ALT+F

*Keyboard shortcuts for the File menu commands*

Option	Keyboard shortcut
Session New...	CTRL+SHIFT+N
Session Open...	CTRL+O
Session Close	none available
Session Save...	CTRL+S
Session Save As...	none available
Soft Reset	CTRL+F2
Hard Reset	ALT+F2
Load Program File...	CTRL+SHIFT+C
Restart	CTRL+SHIFT+R
Save Binary...	none available
Load Binary...	none available
Print...	CTRL+P
Print Setup...	none available
Exit	ALT+F4



**Edit menu ALT+E**

*Keyboard shortcuts for the Edit menu commands*

Option	Keyboard shortcut
Undo	CTRL+Z
Redo	CTRL+Y
Cut	CTRL+X
Copy	CTRL+C
Paste	CTRL+V
Find...	CTRL+F
Find Next	F3
Replace...	none available
Go To...	CTRL+G

**View menu ALT+V**

*Keyboard shortcuts for the View menu commands*

Option	Keyboard shortcut
Toolbar...	none available
Status Bar	none available
Properties...	none available

## Appendix A: Frequent operations

---

### Project menu ALT+P

*Keyboard shortcuts for the Project menu commands*

Option	Keyboard shortcut
Setup Project...	none available
Setup Editor...	none available
Make	CTRL+M
Stop Make	none available
Edit Source Path...	none available
Set FileServer Root Directory...	none available

### Debug menu ALT+D

*Keyboard shortcuts for the Debug menu commands*

Option	Keyboard shortcut
Execution	none available
Run All	CTRL+F9
Stop All	none available
Run/Stop	F9
Run to Address...	SHIFT+F9
Run to Cursor	ALT+F9
Single Step	F7
Forced Step Into	none available
Step Over	F8
Step Out	CTRL+F8
Unstep	CTRL+F7
Enable Animated Step Run	none available

Option	Keyboard shortcut
Step Run In	SHIFT+F7
Step Run Out	SHIFT+F8
Step Run	ALT+F7
Step Run Until...	ALT+F8
Restart	CTRL+SHIFT+R
Breakpoints	none available
Toggle Breakpoint	F5
Enable Breakpoint	none available
Disable Breakpoint	none available
Configure Breakpoint(s)...	CTRL+F5
Reset all Breakpoints	ALT+F5
Enable all Breakpoints	CTRL+SHIFT+F5
Disable all Breakpoints	CTRL+ALT+F5
Remove all Breakpoints	SHIFT+F5
Set Cursor to PC	CTRL+SHIFT+P
Set PC to Cursor	CTRL+ALT+P
Goto Address...	CTRL+G
Configure DA Settings...	none available

## Appendix A: Frequent operations

---

### Region menu ALT+R

*Keyboard shortcuts for the Region menu commands*

Option	Keyboard shortcut
Split Left	CTRL+SHIFT+LEFT ARROW
Split Right	CTRL+SHIFT+RIGHT ARROW
Split Up	CTRL+SHIFT+UP ARROW
Split Down	CTRL+SHIFT+DOWN ARROW
Delete	CTRL+D
Type	none available
Disassembly	ALT+1
Locals	ALT+3
Memory	ALT+4
Register	ALT+5
Source	ALT+6
Watch	ALT+7
Edit	ALT+8
Call Stack	ALT+9
Update all regions now	CTRL+U
Stop all region updates	CTRL+SHIFT+U

## Tools menu

*Keyboard shortcuts for the Tools menu commands*

Option	Keyboard shortcut
Simulate Processor	CTRL+ALT+Z
Profiler	CTRL+ALT+X
Workshop	none available
Disable Updates	none available
Open Door	none available
Close Door	none available
Switch to Emulator	none available
Switch to GD-ROM	none available
Nudge	none available
Hard Errors On	none available
Scripts	none available
Run Script...	none available
Customize	none available
Keyboard...	none available
Tools...	none available
Scripts...	none available
Options...	none available

## Appendix A: Frequent operations

---

### Window menu ALT+W

*Keyboard shortcuts for the Window menu commands*

Option	Keyboard shortcut
New Window	CTRL+N
Cascade	none available
Tile	none available
Arrange Icons	none available
Close All Windows	none available

### Help menu ALT+H, or F1

*Keyboard shortcuts for the Help menu commands*

Option	Keyboard shortcut
Help Topics...	F1
Keyboard	none available
About CodeScape...	none available

## New Windows

*Keyboard shortcuts for creating a new window of a specific region type*

Option	Keyboard shortcut
Disassembly	ALT+1
Locals	ALT+3
Memory	ALT+4
Register	ALT+5
Source	ALT+6
Watch	ALT+7
Edit	ALT+8
Call Stack	ALT+9

## miscellaneous commands options

*Keyboard shortcuts for miscellany*

Option	Keyboard shortcut
Toggle Window Headers	CTRL+ALT+H
Evaluate Expression	CTRL+E
Cycle Radix	CTRL+H

## Appendix A: Frequent operations

---

### Profiler options

*Keyboard shortcuts for the Profiler commands*

Option	Keyboard shortcut
Save Profile	none available
Load Profile	none available
Enable Profiler	none available
One Pass Between Breakpoints	none available
Remove All Profiler Breakpoints	none available
Trace Tree Profile Display	none available
Function Profile Display	none available
Function Profile Filter	none available
Untag All	none available
Sort	none available
Source Display	none available
Disassembly Display	none available
Rename Function...	none available
Profiler Display Setup...	none available
Setup...	none available



**Disassembly region**

*Keyboard shortcuts for the Disassembly region commands*

Option	Keyboard shortcut
Synchronize Cursor	none available
Source Annotation	CTRL+SHIFT+A
Show Address	CTRL+A
Show Labels	CTRL+B
Show Opcode Words	CTRL+W
Show Hexadecimal	CTRL+H
Show Uppercase	CTRL+L
Show Symbols	CTRL+Y
Show EAs & Lits.	CTRL+I
Goto Source File...	none available
View As (Source, Disassembly, or Both)	none available
Evaluate	none available
Tools	none available
Find...	CTRL+F
Find Next	F3
Disassemble to File...	none available

## Appendix A: Frequent operations

---

### Source region

*Keyboard shortcuts for the Source region commands*

Option	Keyboard shortcut
Show Address	CTRL+A
Show Line Nos.	CTRL+L
Tools	none available
Find...	CTRL+F
Find Next	F3
Tab Width...	CTRL+T
Syntax Highlighting	none available

### Call Stack region

*Keyboard shortcuts for the Call Stack region commands*

Option	Keyboard shortcut
Show Parameter Names	none available
Show Parameter Types	none available
Show Parameter Values	none available
Show Parameter Registers	none available
Show Octal	none available
Show Decimal	none available
Show Hexadecimal	none available

**Editor region**

*Keyboard shortcuts for the Editor region commands*

Option	Keyboard shortcut
New	none available
Open...	none available
Save	none available
Save As...	none available
Tabs...	none available
Find...	CTRL+F
Find Next	F3
Replace...	CTRL+H
Go To	CTRL+G
Bookmarks	none available
Toggle	CTRL+B
Next	F2
Previous	none available
Delete All	none available

## Appendix A: Frequent operations

---

### Editor Keys

*Keyboard shortcuts for the Editor Keys commands*

Option	Keyboard shortcut
Select	none available
Select All	CTRL+A
Select Up	SHIFT+UP ARROW
Select Down	SHIFT+DOWN ARROW
Select Start	SHIFT+HOME
Select End	SHIFT+END
Select To End Of File	CTRL+SHIFT+END
Select To Start Of File	CTRL+SHIFT+HOME
Select Page Down	SHIFT+PAGE DOWN
Select Page Up	SHIFT+PAGE UP
Select Left	SHIFT+LEFT ARROW
Select Right	SHIFT+RIGHT ARROW
Select Word Left	CTRL+SHIFT+LEFT ARROW
Select Word Right	CTRL+SHIFT+RIGHT ARROW
Cursor Movement	none available
Move Down	DOWN
Move Up	UP
Move End	END
Move Home	HOME
Move Down A Page	PAGE DOWN
Move Up A Page	PAGE UP

Option	Keyboard shortcut
Move To Top	CTRL+HOME
Move To Bottom	CTRL+END
Move Left	LEFT ARROW
Move Right	RIGHT ARROW
Move Word Left	CTRL+LEFT ARROW
Move Word Right	CTRL+RIGHT ARROW
Backspace	BACKSPACE
Delete	DELETE
Toggle Insert/Overwrite	INSERT
Tab	TAB
Back Tab	SHIFT+TAB

## Appendix A: Frequent operations

---

### Local Watch region

*Keyboard shortcuts for the Local Watch region commands*

Option	Keyboard shortcut
Delete	DELETE
Open	RIGHT ARROW
Close	LEFT ARROW
Keep in View	CTRL+ALT+V
Show Octal	none available
Show Decimal	none available
Show Hexadecimal	none available
Edit Local Value...	CTRL+ALT+E
Highlight Changes	none available
Cache Expanded Symbols	none available

## Memory region

*Keyboard shortcuts for the Memory region commands*

Option	Keyboard shortcut
Display Bytes	CTRL+B
Display Words	CTRL+W
Display Longs	CTRL+L
Display Quadwords	CTRL+Q
Display ASCII	CTRL+A
Highlight Changes	none available
Set Bytes Per Line...	CTRL+SHIFT+L
Edit ASCII	CTRL+ALT+A
Edit Memory Value...	CTRL+ALT+E
Follow Pointer	CTRL+T
Write Protect	CTRL+ALT+W
Tools	none available
Find...	CTRL+F
Find Next	F3
Fill...	none available
Hex Dump to File...	none available

## Appendix A: Frequent operations

---

### Register region

*Keyboard shortcuts for the Register region commands*

Option	Keyboard shortcut
Increment Register	NUM +
Decrement Register	NUM -
Change Inc/Dec Value...	none available
Write Protect	CTRL+ALT+W
Edit Register...	CTRL+ALT+E
Column Format	none available
2 Columns	CTRL+2
4 Columns	CTRL+4
Auto	CTRL+0
Show Banked Registers	CTRL+B
Show Float Registers	CTRL+L
Show Contents at SEA/DEA	none available
Show Float Registers As Hexadecimal	none available
Tools	none available
Save Registers	none available
Restore Registers	none available



## Watch region

*Keyboard shortcuts for the Watch region commands*

Option	Keyboard shortcut
Delete	DELETE
Open	RIGHT ARROW
Close	LEFT ARROW
Insert	CTRL+I
Append	CTRL+A
Show Octal	none available
Show Decimal	none available
Show Hexadecimal	none available
Edit Watch Value...	CTRL+ALT+E

## Symbols

*Keyboard shortcuts for the Symbol Completion dialog box*

Option	Keyboard shortcut
Symbol Complete	ALT+S
Choose Global/Static	ALT+G

## Build

*Keyboard shortcuts for the Project Build commands*

Option	Keyboard shortcut
Next Error	F4
Previous Error	SHIFT+F4

## Appendix A: Frequent operations

---

### Simulator

*Keyboard shortcuts for the Simulator commands*

Option	Keyboard shortcut
Highlight Cache Misses	none available
Highlight Pipeline Stalls	none available
Show Stall Type	none available
Source/Disassembly Tracking	none available
Print...	none available
Save to file...	none available

### Target Settings

*Keyboard shortcuts for configuring the target processor*

Option	Keyboard shortcut
Configure Processor	none available

**ToolBars**

*Keyboard shortcuts for displaying the toolbars*

Option	Keyboard shortcut
Breakpoint	none available
Debug	none available
Processor Combo	none available
Input / Output	none available
Region	none available
Region Combo	none available
Splitter	none available
Standard	none available
Target	none available
Target Combo	none available
Editor	none available
Workshop	none available

## ***Appendix A: Frequent operations***

---

## Script commands in CodeScape Version 2.2.0 Build 111

### Write scripts to automate tasks

CodeScape's script commands let you run Microsoft® JScript™ and VBScript macro scripts to automate routine tasks.

CodeScape's script commands are demonstrated in the example JScript and VBScript files included in this document. Use the functions available in either script language to add your own commands.

For details about using JScript and VBScript connect to the scripting area on the Microsoft Developer Network at <http://msdn.microsoft.com/scripting>

#### CodeScape's script commands

---

**Note:** Some scripting engines reserve the use of Write, use CodeScape's WriteMessage function when you need the Write function.

---

Description	Syntax
Load the specified program file. This command uses the file path as a parameter and returns 1 if the file is loaded, else 0.	LoadProgramFile( path and filename)
Reset the target processor with a hard reset.	HardReset()
Reset the target processor with a soft reset.	SoftReset()
Run the target processor.	Run()
Write a message string to the script window.	WriteMessage( string Message)
Set the specified register to the given value.	WriteRegister( Register value, Numeric value)
Get the value held in the specified register.	RegisterValue ReadRegister( RegisterName )
Load a binary file from the specified location.	LoadBinaryFile( Path and filename, Numeric binary location)
Set a code breakpoint at the specified address.	SetBreakpoint( Numeric address)
Clear all breakpoints.	ClearAllBreakpoints()
Remove the breakpoint from the specified address.	RemoveBreakpoint( Numeric address)
Read a byte from the specified area of memory.	ReadByte( Numeric address)
Read a word from the specified area of memory.	ReadWord( Numeric address)
Read a long from the specified area of memory.	ReadLong( Numeric address)
Write a byte from the specified area of memory.	WriteByte(Numeric address, Numeric value)

Description	Syntax
Write a word from the specified area of memory.	WriteWord(Numeric address, Numeric value)
Write a long from the specified area of memory.	WriteLong(Numeric address, Numeric value)
Return a specific parameter.	GetParam( short param)
Returns the number of parameters passed to the script.	GetParamCount()
Return 1 if running, 0 if not running.	IsRunning()
Specify the events saved in the Trace history:  <div style="display: flex; justify-content: space-between;"> <span><b>Events</b></span> <span><b>Setting</b></span> </div> Log exceptions, interrupts, and rte      8 Log subroutines, bsr, bsrf, jsr, rts      4 Log branches, bf, bt, bf/s, bt/s, bra, braf,jmp    2	ConfigureTraceHistory( numeric Setting, boolean Enable)
Display the current history in the script's window in this format:  <pre>Source      Destination 0x0c010356  0x0c0103aa   rts 0x0c0101e6  0x0c010350   rts 0x0c0100e6  0x0c010128   bra    \$0c010128 0x0c01034c  0x0c010028   bsr    BigTest 0x0c0103a6  0x0c010334   bsr    struct_test 0x0c010280  0x0c0103a0   rts 0x0c01039c  0x0c010214   bsr    BitFieldTest</pre>	DisplayTraceHistory()
Clear the script output window.	ClearDisplay()
Create a breakpoint of the given type at the address. Returns a breakpoint identifier on success, otherwise 0. The breakpoint identifier is used in subsequent operations on the breakpoint.  <div style="display: flex; justify-content: space-between;"> <span><b>Breakpoint</b></span> <span><b>Type</b></span> </div> Code                  0 Watch                 1 Simulator or Start    2 Profiler start        3 Profiler stop         4	CreateBreakpoint( Type, Address)
Enable or disable the breakpoint: identifier: the breakpoint identifier. enable: 1 to enable; 0 to disable.	EnableBreakpoint( identifier, boolean enable)
Enable or disable the specified breakpoint action: enable: 1 to enable , 0 otherwise. identifier: the breakpoint identifier.  <div style="display: flex; justify-content: space-between;"> <span><b>Action</b></span> <span><b>Value</b></span> </div> Halt breakpoint when hit.                  0 Remove breakpoint after being hit.        1 Display a message box prompt when hit    2 Beep when hit                                3	SetBreakpointActions( identifier, numeric action, boolean enable)
Set a log expression for the breakpoint specified by the breakpoint identifier: breakpoint identifier: the breakpoint identifier. expression: the log expression. logType: false to always log or true to log when conditions match.	SetBreakpointLog( breakpoint identifier, string expression, boolean logType)

Description	Syntax																								
Attach a script to a breakpoint: identifier: the breakpoint identifier. script path: the file path for the script script type: 0 for JScript and1 for VBScript script arguments: string holding the script's arguments prompt: 1 to request arguments when the breakpoint triggers, 0 otherwise.	SetBreakpointScript( identifer, string script path, numeric script type, string script arguments, boolean prompt)																								
Set the conditional expression for the breakpoint: identifier: the breakpoint identifier. expression: a string representing the condition. expression type: 0 for C; non-zero for assembly trigger count: the number of hits before breakpoint actions are performed. incOnTrue: false to always increment the trigger count; true to increment the trigger count only when conditions are true. breakWhen: false to break when the trigger reaches 0 or condition is true; true to break when trigger reaches zero and the condition is true.	SetBreakpointCondition( identifier, string expression, numeric expression type, numeric trigger count, boolean incOnTrue, boolean breakWhen)																								
Set the parameters for a watch breakpoint: identifier: the breakpoint identifier. incDataCondition: include a data condition. dataCondition: expression specifying the data condition. expressionType: the type of the specified expression. accessSize: the access size e.g. byte, word, or long etc. accessType: the type of access (read, write, or both).  <table><tr><td><b>Size</b></td><td><b>Value</b></td><td><b>Type</b></td><td><b>Value</b></td></tr><tr><td>Any</td><td>0</td><td>Read</td><td>1</td></tr><tr><td>Byte</td><td>1</td><td>Write</td><td>2</td></tr><tr><td>Word</td><td>2</td><td>Read or Write</td><td>3</td></tr><tr><td>Long</td><td>4</td><td></td><td></td></tr><tr><td>Quad</td><td>8</td><td></td><td></td></tr></table>	<b>Size</b>	<b>Value</b>	<b>Type</b>	<b>Value</b>	Any	0	Read	1	Byte	1	Write	2	Word	2	Read or Write	3	Long	4			Quad	8			BOOL SetWatchBreakpointParameters( Identifier, Boolean incDataCondition, string dataCondition, numeric expressionType, numeric accessSize, numeric accessType)
<b>Size</b>	<b>Value</b>	<b>Type</b>	<b>Value</b>																						
Any	0	Read	1																						
Byte	1	Write	2																						
Word	2	Read or Write	3																						
Long	4																								
Quad	8																								
Select a location mask for the breakpoint:  <table><tr><td><b>Mask</b></td><td><b>Value</b></td></tr><tr><td>No bits masked</td><td>1</td></tr><tr><td>Lower 10 bits</td><td>2</td></tr><tr><td>Lower 12 bits</td><td>3</td></tr><tr><td>Lower 16 bits</td><td>4</td></tr><tr><td>Lower 20 bits</td><td>5</td></tr><tr><td>All bits</td><td>6</td></tr></table>	<b>Mask</b>	<b>Value</b>	No bits masked	1	Lower 10 bits	2	Lower 12 bits	3	Lower 16 bits	4	Lower 20 bits	5	All bits	6	SetBreakpointLocationMask( breakID, maskSelect)										
<b>Mask</b>	<b>Value</b>																								
No bits masked	1																								
Lower 10 bits	2																								
Lower 12 bits	3																								
Lower 16 bits	4																								
Lower 20 bits	5																								
All bits	6																								
Set the data mask for a watch breakpoint	SetBreakpointDataMask( identifier, mask)																								

You can express Numeric values and Numeric addresses as:

- A number, for example, 124 or 3.1415926  
-OR-
- A string, for example, "124"  
-OR-
- Hexadecimal in a string, for example, "0xabcdef"  
-OR-
- A symbol, for example, "main" or "main + 0xabc"

---

**Note:** Registers are only passed as strings, for example, "pc", "fr0", "r0".

---

---

**Note:** Currently, scripts only support debugging a single target processor. When you run a script it automatically uses the selected target processor.

---

---

**Note:** A script that contains an infinite loop will cause CodeScape to lock-up.

---



**Example VBScript**

```
' This script does not do anything useful other than demonstrate the functions available
```

```
ClearDisplay
HardReset
SoftReset
DisplayParameters
LoadSomeBinary
LoadProgramFile( "d:\\projects\\maketest\\hello.elf" )
SetBreakpoint( "add_fn" )
ConfigureTraceHistory TH_LOGEXCEPT + TH_LOGSUB, true
Dim Running
Running = 1
Do
    Running = IsRunning
Loop Until Running = 0
DisplayTraceHistory
ReadSomeRegisters
WriteSomeRegisters
ReadSomeMemory
WriteSomeMemory
ReadSomeMemory
ClearAllBreakpoints
CreateCodeBP
ClearAllBreakpoints
CreateWatchBP
WriteMessage( "Script complete. Removing all breakpoints." )
ClearAllBreakpoints
```

```
'
'      Breakpoint types
'
BPTYPE_CODE          =      0
BPTYPE_WATCH         =      1
BPTYPE_SIMSTART      =      2
BPTYPE_PROFSTART     =      3
BPTYPE_PROFSTOP      =      4

'
'      Breakpoint Actions
'
BPACTION_HALT        =      0
BPACTION_ONESHOT     =      1
BPACTION_PROMPT      =      2
BPACTION_BEEP        =      3

'
'      Breakpoint Script Types
'
BPSCRIPT_JSCRIPT     =      0
BPSCRIPT_VBSCRIPT    =      1

'
'      Breakpoint expression types
'
BPEXPR_C              =      0
BPEXPR_ASSEMBLY      =      1

'
'      Breakpoint address masks
'

BPLOCMASK_NONE        =      1
BPLOCMASK_LOW10       =      2
BPLOCMASK_LOW12       =      3
BPLOCMASK_LOW16       =      4
BPLOCMASK_LOW20       =      5
BPLOCMASK_ALL         =      6
```

```

'
'      Breakpoint access sizes
'

BPACCESSSIZE_ANY      =      0
BPACCESSSIZE_BYTE     =      1
BPACCESSSIZE_WORD     =      2
BPACCESSSIZE_LONG     =      4
BPACCESSSIZE_QUAD     =      8

'
'      Breakpoint access types
'

BPACCESSTYPE_READ      =      1
BPACCESSTYPE_WRITE    =      2
BPACCESSTYPE_RW       =      3

'
'      Trace history configuration options
'

TH_LOGEXCEPT =      8
TH_LOGSUB      =      4
TH_LOGBRANCH  =      2

'
'      Create a breakpoint on the 1K aligned block of memory that
'      the symbol main resides in.
'

Sub CreateCodeBP()

    Dim breakID

    breakID      =      CreateBreakpoint( BPTYPE_CODE, "main" )

    SetBreakpointAction breakID, BPACTION_HALT, true

    SetBreakpointAction breakID, BPACTION_ONESHOT, false

    SetBreakpointAction breakID, BPACTION_PROMPT, false
    SetBreakpointAction breakID, BPACTION_BEEP, true
    SetBreakpointScript breakID, "e:\\projects\\codescape\\debugs\\testscript.js",
BPSCRIPT_JSCRIPT, "arg1 arg2 arg3", false
    SetBreakpointLog breakID, "Hello John", BPEXPR_C
    SetBreakpointLocationMask breakID, BPLOCMASK_LOW10
    setBreakpointCondition breakID, "index == 375", BPEXPR_C, 37, true, true
End Sub

Sub CreateWatchBP()
    breakID      = CreateBreakpoint( BPTYPE_WATCH, "main" )
    SetWatchBreakpointParameters breakID, true, "14", BPEXPR_C, BPACCESSSIZE_BYTE,
BPACCESSTYPE_WRITE
End Sub

Sub WriteSomeRegisters()
    WriteRegister "fr0", 3.14159
    WriteRegister "r0", "0xabcdef"
    WriteRegister "pc", "main + 0x30"
End Sub

Sub ReadSomeRegisters()
    WriteMessage( "Value of pc = " & ReadRegister( "pc" ) )
    WriteMessage( "Value of r0 = " & ReadRegister( "r0" ) )
End Sub

```

**Cross Products Company Confidential**

```
Sub LoadSomeBinary()  
    LoadBinaryFile "d:\\projects\\codescape\\satmon.bin", "201392128"  
    LoadBinaryFile "d:\\projects\\codescape\\satmon.bin", 201392128  
    LoadBinaryFile "d:\\projects\\codescape\\satmon.bin", "0xc010000"  
    LoadBinaryFile "d:\\projects\\codescape\\satmon.bin", "main"  
End Sub  
  
Sub DisplayParameters()  
    NumParams = GetParamCount  
    WriteMessage( "Number of parameters = " & NumParams )  
    For i = 1 To NumParams  
        WriteMessage( "Parameter " & i & " = " & GetParam( i - 1 ) )  
    Next  
End Sub  
  
Sub ReadSomeMemory()  
    WriteMessage( "Byte at main = " & ReadByte( "main" ) )  
    WriteMessage( "Word at main + 4 = " & ReadWord( "main + 4" ) )  
    WriteMessage( "Long at main + 8 = " & ReadLong( "main + 8" ) )  
End Sub  
  
Sub WriteSomeMemory()  
    WriteByte "main", 255  
    WriteWord "main + 4", "0xabcd"  
    WriteLong "main + 8", "0xfedcba"  
End Sub
```

**Example JScript**

```

//      Note: this script does not do anything useful. It just demonstrates the current
//      script commands and how they can be called.
//
//      Breakpoint types
//
BPTYPE_CODE           =      0;
BPTYPE_WATCH          =      1;
BPTYPE_SIMSTART       =      2;
BPTYPE_PROFSTART      =      3;
BPTYPE_PROFSTOP       =      4;

//
//      Breakpoint Actions
//
BPACTION_HALT         =      0;
BPACTION_ONESHOT      =      1;
BPACTION_PROMPT       =      2;
BPACTION_BEEP         =      3;

//
//      Breakpoint Script Types
//
BPSCRIPT_JSCRIPT      =      0;
BPSCRIPT_VBSCRIPT     =      1;

//
//      Breakpoint expression types
//
BPEXPR_C              =      0;
BPEXPR_ASSEMBLY       =      1;

//
//      Breakpoint address masks
//
BPLOCMASK_NONE        =      1;
BPLOCMASK_LOW10       =      2;
BPLOCMASK_LOW12       =      3;
BPLOCMASK_LOW16       =      4;
BPLOCMASK_LOW20       =      5;
BPLOCMASK_ALL         =      6;

//
//      Breakpoint access sizes
//
BPACCESSSIZE_ANY      =      0;
BPACCESSSIZE_BYTE     =      1;
BPACCESSSIZE_WORD     =      2;
BPACCESSSIZE_LONG     =      4;
BPACCESSSIZE_QUAD     =      8;

//
//      Breakpoint access types
//
BPACCESSTYPE_READ     =      1;
BPACCESSTYPE_WRITE    =      2;
BPACCESSTYPE_RW       =      3;

//
//      Trace history configuration options
//
TH_LOGEXCEPT        =      8;
TH_LOGSUB             =      4;
TH_LOGBRANCH          =      2;

```

```
//
//      Create breakpoint on the 1k aligned block of memory that the symbol main resides in
//
function CreateCodeBP()
{
    breakID      =      CreateBreakpoint( BPTYPE_CODE, "main" );
    SetBreakpointAction( breakID, BPACTIION_HALT, true );
    SetBreakpointAction( breakID, BPACTIION_ONESHOT, false );
    SetBreakpointAction( breakID, BPACTIION_PROMPT, false );
    SetBreakpointAction( breakID, BPACTIION_BEEP, true );
    SetBreakpointScript( breakID, "e:\\projects\\codescape\\debugs\\testscript.js",
BPSCRIPT_JSCRIPT, "arg1 arg2 arg3", false );
    SetBreakpointLog( breakID, "Hello John", BPEXPR_C );
    SetBreakpointLocationMask( breakID, BPLOCMASK_LOW10 );
    setBreakpointCondition( breakID, "index == 375", BPEXPR_C, 37, true, true );
}

function CreateWatchBP()
{
    breakID      =      CreateBreakpoint( BPTYPE_WATCH, "main" );
    SetWatchBreakpointParameters( breakID, true, "14", BPEXPR_C, BPACCESSSIZE_BYTE,
BPACCESSTYPE_WRITE );
}

function WriteSomeRegisters()
{
    WriteRegister( "fr0", 3.14159 );
    WriteRegister( "r0", "0xabcdef" );
    WriteRegister( "pc", "main + 0x30" );
}

function ReadSomeRegisters()
{
    WriteMessage( "Value of pc = " + ReadRegister( "pc" ) );
    WriteMessage( "Value of r0 = " + ReadRegister( "r0" ) );
}

function LoadSomeBinary()
{
    LoadBinaryFile( "d:\\projects\\codescape\\satmon.bin", "201392128" );
    LoadBinaryFile( "d:\\projects\\codescape\\satmon.bin", 201392128 );
    LoadBinaryFile( "d:\\projects\\codescape\\satmon.bin", "0xc010000" );
    LoadBinaryFile( "d:\\projects\\codescape\\satmon.bin", "main" );
}

function DisplayParameters()
{
    NumParams      =      GetParamCount()
    WriteMessage( "Number of parameters = " + NumParams );
    for( i = 0; i < NumParams; i++ )
    {
        WriteMessage( "Parameter " + i + " = " + GetParam( i ) )
    }
}

function ReadSomeMemory()
{
    WriteMessage( "Byte at main = " + ReadByte( "main" ) );
    WriteMessage( "Word at main + 4 = " + ReadWord( "main + 4" ) );
    WriteMessage( "Long at main + 8 = " + ReadLong( "main + 8" ) );
}

function WriteSomeMemory()
{
    WriteByte( "main", 255 );
    WriteWord( "main + 4", "0xabcd" );
    WriteLong( "main + 8", "0xfedcba" );
}
```

```
ClearDisplay();
HardReset();
SoftReset();
DisplayParameters();
LoadSomeBinary();
LoadProgramFile( "d:\\projects\\maketest\\hello.elf" );
SetBreakpoint( "add_fn" );
ConfigureTraceHistory( TH_LOGEXCEPT + TH_LOGSUB, true );
Run();
while( IsRunning() != 0 )
{
    ;
}
DisplayTraceHistory();
ReadSomeRegisters();
WriteSomeRegisters();
ReadSomeMemory();
WriteSomeMemory();
ReadSomeMemory();
ClearAllBreakpoints();
CreateCodeBP();
ClearAllBreakpoints();
CreateWatchBP();
WriteMessage( "Script complete. Removing all breakpoints." );
ClearAllBreakpoints();
```

## Using scripts

When you run a script the Input / Output window appears automatically and displays the Script tab with all messages generated by the current script.

To open the Input / Output window without running a script:

- Click View, Toolbar, then select the Input / Output check-box and click OK.

---

**Note:** You can dock the Input / Output window at the top and bottom of the main window, or leave it free floating.

---

### *The shortcut menu on the Script tab*

Click:	To:
Run Script	Select and run a script.
Clear	Clear the contents of the Script tab.
User Scripts	This option appears in gray until you add a script to the menu. When you add a script its name appears on the menu.
Allow Docking	Toggle docking for the window on or off.
Hide	Hide the window.

### Add a script to the menu

When you add a script its name appears on the menu bar, and on the Script tab shortcut menu. You can add up to ten script files to run from either the menu bar, or the shortcut menu.

- 1 Click Tools, select Customize, then click Scripts...  
The Customize dialog box appears.
- 2 Click Add.
- 3 In the Menu Text box, enter the script name to display on menu.  
To remove an entry from the Menu Text box, select the script, then click Remove.
- 4 In the Menu Contents box, highlight the name of the script.
- 5 In the Script box, enter the path location and script file name.
- 6 Select either JScript, or VBScript to specify the script file type.
- 7 Do one of the following:
  - In the Arguments text box, enter any arguments to be passed to the script. Click OK.
  - OR–
  - Select the Prompt for arguments check-box.

---

**Note:** Select a command in the Menu Contents box, then Use Move Up and Move Down to set where it appears on the Tools menu.

---



---

**Note:** To assign a keyboard shortcut to the script click Tools, select Customize then click Keyboard...

---

### Run a script

- Click Tools, then select Scripts and click a script in the list.
- OR-
- On the Input / Output window, right-click on the Scripts tab, then click a script in the list.

---

**Note:** Currently, scripts only support debugging a single target processor. When you run a script it automatically uses the selected target processor.

---

---

**Note:** A script that contains an infinite loop will cause CodeScape to lock-up.

---



# **LibCross fileserver**

for CodeScape v2.2.0 build 114

## LibCross fileserver

The LibCross fileserver provides low level routines that interface CodeScape with the standard C run-time library (*libc.a*). The fileserver supports the following functions:

```
int debug_open (const char *filename, int flags);
int debug_close (int file);
int debug_read(int file, char *ptr, int len);
int debug_write (int file, char *ptr, int len);
int debug_lseek(int file, int offset, int origin);

char * debug_getcwd(char *buffer, int maxlen);
int debug_chdir(const char *dirname);
int debug_mkdir(const char *dirname);
int debug_rmdir(const char *dirname);

int debug_findfirst(const char *filespec, struct SNASM_finddata_t *fileinfo);
int debug_findnext(int handle, struct SNASM_finddata_t *fileinfo);
int debug_findclose(int handle);

int _ASSERT(int nFlag);

int dedug_printf(char *format, ...);
```

---

**Note:** The header file **usrsnasm.h** has information on using the fileserver functions. It defines all functions and custom data types such as struct SNASM\_finddata\_t.

---

---

**Note:** If the fileserver returns an error errno describes the problem. For information about errno refer to your C run-time library documentation.

---

### This release includes:

- .\libcrs - contains source, object files for the transport functions. The source of wrapper functions for Hitachi system calls.
- .\sample - contain a demonstration program 'sample.elf'.

### Using the fileserver with the Hitachi SHC compiler

To use the fileserver with the Hitachi SHC compiler you need wrapper functions for the system calls open(),close(),read(), write(), lseek(). The source code for these wrapper functions is included in this release.

---

**Note:** Do not transfer more than 32K in any SINGLE read or write command as not all communications are buffered by the fileserver transport functions.

---

## Fileserver functions

### Open a file

### Required header

```
int debug_open (const char *filename, int flags);
```

```
#include <usrsnasm.h>
```

### Return Value

Returns a file handle for an open file. If the return value is `-1` an error occurred, refer to ***errno*** for one of the following:

The <b>errno</b> setting:	Means that the file cannot be opened because:
SNASM_EACCESS	It is read-only; or it is not a shared resource; or the path or filename are incorrect.
SNASM_EEXIST	The filename already exists.
SNASM_EINVAL	An invalid flags argument is defined.
SNASM_EMFILE	No file handles are available, close one or more files and try again.
SNASM_ENOENT	File or path not found.

### Parameters

filename    Name of file to open.  
 flags        Open flags for type of operations desired.

### Remarks

The *flags* parameter can be a combination of the following definitions defined in `<sn_fcntl.h>`.

SNASM_O_RDONLY	open for read only
SNASM_O_WRONLY	open for write only
SNASM_O_RDWR	open for read and write
SNASM_O_APPEND	writes done at end of file
SNASM_O_CREAT	create new file
SNASM_O_TRUNC	truncate existing file
SNASM_O_NOINHERIT	file is not inherited by child process
SNASM_O_TEXT	text file
SNASM_O_BINARY	binary file
SNASM_O_EXCL	exclusive open

SNASM\_O\_BINARY and SNASM\_O\_TEXT are essential when opening the file if the host is an IBM PC or Compatible device.

<b>Note:</b>	Within the <i>open</i> wrapper command for Hitachi the <i>flags</i> parameter is translated from machine specific to a compiler independent format for translation transfer to the host. For example, O_BINARY will be converted to SNASM_O_BINARY.
--------------	---

## Close a file

```
int debug_close ( int file);
```

## Required header

```
#include <usrnasm.h>
```

## Return Value

*debug\_close* returns 0 if the file closed successfully. If the return value is -1 an error occurred, refer to **errno** for the following:

The errno setting:	Means that the file cannot be closed because:
SNASM_EBADF	The file handle is invalid.

## Parameters

file      Handle returned by *debug\_open* to the file.

## Remarks

CodeScape will close all open file handles when either the target is reset or when the CodeScape application is closed.

## Read data from a file

```
int debug_read( int file, char *ptr, int len)
```

## Required header

```
#include <usrsnasm.h>
```

## Return Value

On success *debug\_read* returns the number of bytes read. If the function tries to read at end of file, it returns 0. If the return value is `-1` an error occurred, refer to ***errno*** for the following:

The <b>errno</b> setting:	Means that the data cannot be read because:
SNASM_EBADF	The file handle is invalid; or the file is not open for reading; or the file is locked.

## Parameters

file	Handle to the file.
ptr	Pointer to buffer where read data is to be stored.
len	Maximum number of bytes.

## Remarks

The *debug\_read* operation occurs from the position of the file pointer. After a successful *debug\_read*, the file position is at the return value number of bytes along the file.

Use *debug\_lseek* to move the file position around.

**Write data to a file**

```
int debug_write ( int file, char *ptr, int len);
```

**Required header**

```
#include <usrsnasm.h>
```

**Return Value**

*debug\_write* returns the number of bytes written. If the return value is `-1` an error occurred, refer to **errno** for one of the following:

The errno setting:	Means that:
SNASM_EBADF	The file handle is invalid; or the file is not open for writing.
SNASM_ENOSPC	There is not enough available disk space.

**Parameters**

file	Handle to the file.
ptr	Pointer to buffer where write data is stored.
len	Number of bytes.

**Remarks**

Two channels, *SNASM\_STDOUT* and *SNASM\_STDERR*, are used to display information on the Log tab of CodeScape's Input / Output window by default.

## Move a file to a specific location

## Required header

```
int debug_lseek ( int file, int offset, int origin)
```

```
#include <usrsnasm.h>
```

## Return Value

*debug\_lseek* returns the offset, in bytes, of the new position from the beginning of the file. If the return value is `-1` an error occurred, refer to ***errno*** for one of the following:

The <b>errno</b> setting:	Means that the:
SNASM_EBADF	File handle is invalid.
SNASM_ENIVAL	Origin value is invalid; or the specified location is before the start of the file.

## Parameters

file	Handle to the file.
offset	Number of bytes from origin.
origin	Flag indicating the origin.

## Remarks

The *origin* flag can be any of the following predefined values:

SNASM_SEEK_SET	From start of file position
SNASM_SEEK_CUR	From current position
SNASM_SEEK_END	From end of file

**Get current working directory**

```
char * debug_getcwd ( const char *buffer, int maxlen)
```

**Required header**

```
#include <usrsnasm.h>
```

**Return Value**

*debug\_getcwd* returns a pointer to the buffer. If the return value is NULL an error occurred, refer to **errno** for the following:

The errno setting:	Means that the:
SNASM_ERANGE	Path is longer than <i>maxlen</i> characters.

**Parameters**

buffer	Allocated space in which to store the path.
maxlen	Number of bytes from in buffer.

**Remarks**

The working directory is specified in CodeScape's *Set Fileserver Path...* dialog box.



## Change current working directory

## Required header

```
int debug_chdir ( const char *dirname)
```

```
#include <usrsnasm.h>
```

## Return Value

*debug\_chdir* returns a value of 0. If the return value is -1 an error occurred, refer to ***errno*** for the following:

The <b>errno</b> setting:	Means that the:
SNASM_ENOENT	Specified path could not be found.

## Parameters

dirname      Path of the new working directory.

## Remarks

The directory set in the *dirname* parameter must exist. The function may be used to change the drive and working directory. For example, to change the drive to "C" and the working directory to "window\temp" enter: **`debug_chdir("c:\\windows\\temp");`**

Use "\\ " to describe a single "\" in a C string literal.

The working directory is specified in CodeScape's *Set Fileserver Path...* dialog box.

## Create a new directory

```
int debug_mkdir ( const char *dirname)
```

## Required header

```
#include <usrsnasm.h>
```

## Return Value

*debug\_mkdir* returns a value of 0. If the return value is -1 an error occurred, refer to **errno** for one of the following:

The errno setting:	Means that the directory cannot be created because:
SNASM_EEXISTS	It already exists.
SNASM_ENOENT	The specified path does not exist.

## Parameters

dirname      Path of the new directory.

## Remarks

The function only creates one directory per call.

## Delete a new directory

```
int debug_rmdir ( const char *dirname)
```

## Required header

```
#include <usrsnasm.h>
```

## Return Value

*debug\_rmdir* returns a value of 0. If the return value is `-1` an error occurred, refer to ***errno*** for one of the following:

The <i>errno</i> setting:	Means that the directory cannot be deleted because:
SNASM_EACCESS	It does not exist; or it is not empty; or it is the current working directory; or it is the root directory.
SNASM_ENOENT	The specified path was not found.

## Parameters

dirname      Path of the new directory.

## Remarks

The function deletes the specified directory. The directory must be empty and it cannot be the root directory or the current working directory.

**Information about the first instance of a filename****Required header**

```
int debug_findfirst ( const char *filespec, struct SNASM_finddata_t *
fileinfo )
```

```
#include <usrsnasm.h>
```

**Return Value**

*debug\_findfirst* returns a search handle. If the return value is `-1` an error occurred, refer to **errno** for one of the following:

The errno setting:	Means that the file specification:
SNASM_ENOENT	Is invalid.
SNASM_EINVAL	Could not be found.

**Parameters**

filespec      Target file specification.  
fileinfo      Pointer to structure to hold file specification.

**Remarks**

The function returns information on the first file that matches the file specification. The file specification can contain wildcards, for example, the following command searches for C files in the current working directory:

```
int hSearchHandle = debug_findfirst("*.c", &FileSpecification);
```

The file information structure contains 3 parameters:

```
unsigned long      m_ulSize;                                /* file size                */
unsigned long      m_ulAttributes;                        /* file attributes */
char                m_szFilename[260]; /* file name                */
```

The attributes will be one of the following values:

```
SNASM_A_NORMAL      /* Normal. File can be read or written to without
restriction. */
SNASM_A_RDONLY      /* Read-only. File cannot be opened for writing, and a
file with the same name cannot be created. */
SNASM_A_HIDDEN      /* Hidden file. Not normally seen with the DIR command,
unless the /AH option is used. Returns information about normal files as well
as files with this attribute.*/
SNASM_A_SYSTEM      /* System file. Not normally seen with the DIR command,
unless the /A or /A:S option is used. */
SNASM_A_SUBDIR      /* Subdirectory. */
SNASM_A_ARCH        /* Archive. Set whenever the file is changed, and cleared
by the BACKUP command. */
```

**Information about the next instance of a filename****Required header**

```
int debug_findnext ( int handle, struct SNASM_finddata_t * fileinfo )
```

```
#include <usrsnasm.h>
```

**Return Value**

*debug\_findnext* returns 0. If the return value is -1 an error occurred, refer to **errno** for the following:

The errno setting:	Means that:
SNASM_ENOENT	No more files matched the file specification.

**Parameters**

handle      Search handle supplied by *debug\_findfirst*.  
fileinfo     Pointer to a structure to hold file specification.

**Remarks**

The function returns information on the next file that matches the file specification.

The file information structure contains 3 parameters:

```
unsigned long      m_ulSize;                               /* file size               */
unsigned long      m_ulAttributes;                       /* file attributes */
char               m_szFilename[260]; /* file name               */
```

The attributes field shows one of the following values:

```
SNASM_A_NORMAL       /* Normal. File can be read or written to without
restriction. */
SNASM_A_RDONLY       /* Read-only. File cannot be opened for writing, and a
file with the same name cannot be created. */
SNASM_A_HIDDEN       /* Hidden file. Not normally seen with the DIR command,
unless the /AH option is used. Returns information about normal files as well
as files with this attribute.*/
SNASM_A_SYSTEM       /* System file. Not normally seen with the DIR command,
unless the /A or /A:S option is used. */
SNASM_A_SUBDIR       /* Subdirectory. */
SNASM_A_ARCH       /* Archive. Set whenever the file is changed, and cleared
by the BACKUP command. */
```

## Close a search handle

```
int debug_findclose ( int handle )
```

## Required header

```
#include <usrnasm.h>
```

## Return Value

*debug\_findclose* returns 0. If the return value is  $-1$  an error occurred and the operation failed to close the handle.

## Parameters

handle      Search handle supplied by *debug\_findfirst*.

## Remarks

Free up resources allocated to the file search operations.

**Halt and inform the user**

```
int _ASSERT ( int nFlag )
```

**Required header**

```
#include <usrsnasm.h>
```

**Return Value**

Returns 0.

**Parameters**

nFlag      Test *nFlag*, if expression evaluates to zero an assert is generated on host.

**Remarks**

When a `_ASSERT` occurs and the flag evaluates to zero the host is told. The host prompts for instruction and the `_ASSERT()` encountered dialog box appears, select:

**Yes** to stop the program and tell CodeScape to put the cursor on the `_ASSERT` statement.

**No** to ignore the assert and continue running the program.

**Cancel** to ignore this and all further asserts.

You can set and view the status and control of *ignore all* in CodeScape's *Global Options* dialog box.

Some compilers generate code that cause CodeScape to stop on the instruction following a `_ASSERT`. The sample program supplied includes a macro that ensures that a `_ASSERT` will stop on the line that generated it. The file also shows how all asserts can be removed with a global definition.

```
/*
 * Macro Redefinition of _ASSERT to ASSERT. This is performed to cause the
 * compiler to insert at least one opcode after the jsr _ASSERT has returned
 * it also permits the ASSERT code to be included / removed based on a
 * compiler define.
 */
#ifdef _DEBUG_BUILD_
    /* Since _ASSERT always return zero the expression will only be evaluated once */
    #define ASSERT(X) while(!_ASSERT(X)) { ; }
#else
    #define ASSERT(X)
#endif /* _DEBUG_BUILD_ */
```

## Prints data to the Log tab

## Required header

```
int dedug_printf(char *format, ...);
```

```
#include <usrnasm.h>
```

## Return Value

The return value is the number of characters printed to the Log tab. Returns a negative value if an error occurs.

## Parameters

<i>Format</i>	Format control
<i>Argument</i>	Optional arguments

## Remarks

The function formats and prints data to the Log tab on the Input / Output window.

---

<b>Note:</b>	If arguments follow the <i>format</i> string, the <i>format</i> string must contain argument output format specifications. The <i>format</i> argument consists of ordinary characters, escape sequences, and (if arguments follow <i>format</i> ) format specifications.
--------------	--

---