

---

# *CodeScape*

## *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

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.

Release Version 2.2.0: March 1999; July 1999; October 1999

© 1996 - 2000 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.

Cross Products Ltd  
23 The Calls, Leeds, West Yorkshire, LS2 7EH  
telephone: +44 113 242 9814  
facsimile: +44 113 242 6163  
www.crossprod.co.uk  
email sales: enquiry@crossprod.co.uk  
email support: support@crossprod.co.uk

---

# Contents

<b>About this guide .....</b>	<b>1</b>
The CodeScape software .....	3
<b>Using and configuring the interface .....</b>	<b>5</b>
Menu bar commands .....	5
Customizing shortcut keys .....	8
Toolbars and their uses .....	10
Breakpoint toolbar .....	13
Debug toolbar .....	14
Processor Combo toolbar .....	15
Input/Output window .....	16
Region toolbar .....	17
Region Combo toolbar .....	18
Splitter toolbar .....	19
Standard toolbar .....	20
Target window .....	21
Target Combo toolbar .....	21
Editor toolbar .....	22
Status Bar .....	23
<b>How windows and regions work .....</b>	<b>25</b>
Using windows .....	26
Using regions .....	28
Configuring regions .....	32
Target window .....	34
Target Processor display .....	35
Input/Output window .....	35
Source and Disassembly regions .....	38
Call Stack region .....	46
Watch and Local Watch regions .....	47
Memory region .....	56
Register region .....	65
Edit region .....	73

---

<b>Interacting with target processors .....</b>	<b>79</b>
Connecting to a target processor .....	80
Add files to a project .....	82
Restart a program .....	85
Configure the network settings .....	85
<b>Working with sessions .....</b>	<b>87</b>
<b>Working with projects .....</b>	<b>91</b>
Setting up a project build environment .....	92
Setting up an editor .....	94
<b>Debugging .....</b>	<b>99</b>
Loading the debug stub .....	100
Running and stopping programs .....	102
Stepping into (tracing) code .....	105
<b>Breakpoints .....</b>	<b>111</b>
Configuring breakpoints .....	117
Breakpoint expression format .....	124
<b>Evaluating expressions .....</b>	<b>129</b>
C/C++ expressions .....	132
Assembler expressions .....	134
<b>Writing scripts to automate tasks .....</b>	<b>137</b>
Scripting commands .....	140
<b>Using the command-line .....</b>	<b>159</b>
<b>Configuration file: dali.cfg .....</b>	<b>163</b>
<b>Shortcut keys .....</b>	<b>167</b>

---

# About this guide

---

## Document conventions

---

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

---

---

**CAUTION:**    *Cautions alert you to personal safety risk, system damage, or loss of data.*

---

*Table 1: Typographic conventions in this guide*

Convention	Description
SPACEBAR	Capital letters denote the names of keys on the keyboard, filenames, and extensions.
ALT+F4	If keys must be pressed simultaneously, they are linked with a plus sign (+).
ALT, F, X	If 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.
<b>emphasis</b>	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.

## About each section

**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.

**How windows and regions work** 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** explains the various ways you can debug your project files, including: loading the debug stub, running and stopping programs, and stepping source code.

**Breakpoints** explains how to set and configure watch and data breakpoints, and describes the breakpoint expression format.

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

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

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

**Configuration file: dali.cfg** explains how memory ranges and processor timing information are defined in the configuration file.

**Shortcut keys** describes the keyboard shortcuts available in CodeScape.

## 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 the dongle to be inserted into a parallel port on your computer.

Your computer:

- *Minimum* - an IBM™ PC or compatible with Pentium® 90MHz processor and 32 MB of RAM.
- *Recommended* - an IBM™ PC or compatible with Pentium® II 233MHz processor or above and 128 MB of RAM.

---



# *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, and Regions.

## Menu bar commands

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

The File menu commands 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. When you load a new session, or exit CodeScape, a message appears prompting you to save any changes.

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

The Edit menu commands let you cut and paste in the Editor and perform searches. The Edit menu appears when you open a window, create a region, or load a session.

## **View menu ALT+V**

The View menu commands let you show and hide toolbars and configure regions.

## **Project menu ALT+P**

The commands in the Project menu let you configure and build projects.

## **Debug menu ALT+D**

The Debug menu commands let you control program execution, step code, and use breakpoints. Other options let you:

- View and manage file overlay properties.
- Set the start and end address of a base address range to map one or more associated address ranges from in the Debug Address Mappings dialog box.
- Set the cursor to the PC (program counter) and vice-versa. The default origin is set to the value of the PC.
- Lock the view to an Expression that contains the PC, a register, or memory.

### ***The Debug Address Mappings dialog box***

The options on the dialog box let you set the start and end address of a base address range that you can map one or more associated address ranges from. If you do this, any debug information describing the base address range applies to the associated mapped address ranges instead.

Each associated mapped range can be unmapped when it not required. Removing all associated mapped ranges for a base range hides the debug information for base range unless you:

- Remove the base address range.  
-OR-
- Map one or more new associated mapped ranges to the base range.

Use this feature to hide debug information for an address range by entering it as a base address range without mapping another address range to it.

## **Region menu ALT+R**

The Region menu commands let you split (create new) regions, change a region's type, and update all regions. The Region menu is available when you open or create a window, region, or session.

## Tools menu ALT+T

The Tools menu commands let you run script files and add script files to run from the Tools menu and the shortcut menu on the Scripts tab on the Input / Output window. You can also search for a string in multiple files, set-up target debug support, specify custom shortcut keys, and add programs to run from the Tools menu.

## 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 for arranging multiple windows. You can select and deselect commands to proportionally resize a window and its regions, and loading the last open 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 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. The Region Type menu commands are for changing a region's type. To see the menu:

- Press CTRL+SHIFT+F10.
- OR-
- CTRL+Right-click anywhere in the region.

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 default settings.
- Use the Microsoft Developer Studio shortcut key 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 default settings

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

### Using Microsoft Developer Studio shortcut key 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.*

---

## Toolbars and their uses

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.

*Table 2: 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 and 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, change a region's type.
Splitter toolbar	Split regions using the mouse.
Standard toolbar	Open new windows, and create, open, and save sessions. Cut, copy, and paste data to and from the Windows clipboard. Print the current file and search for a string in multiple files.
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.
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


### View toolbars

- 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

- 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.

## 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 Input/Output 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.

### Breakpoint toolbar



*Table 3: 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

## Debug toolbar

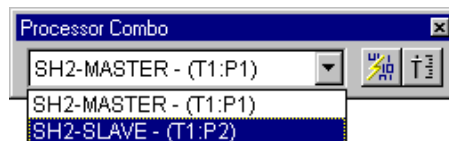


Table 4: Commands on the Debug toolbar

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



To issue this command:	Click:	Press:
Step Run Until		ALT+F8
Set Cursor to PC		CTRL+SHIFT+P
Load Program File		CTRL+SHIFT+C
Unload Program File Info		CTRL+ALT+U
Set PC to Cursor		CTRL+ALT+P
Restart		CTRL+SHIFT+R
Overlay Management		CTRL+ALT+O
Debug Address Mappings		CTRL+SHIFT+M

## 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.

*Table 5: 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.

When the Input / Output window is open and you search for a string in multiple files it automatically displays the:

- **Find in Files tab** with a list of all files containing the search term in the location specified in your search. Double-click an item in the list to examine the associated file.

To search for a string in multiple files:

- Click Tools, Find In Files.

-OR-









- On the Standard toolbar, click .

## Region toolbar

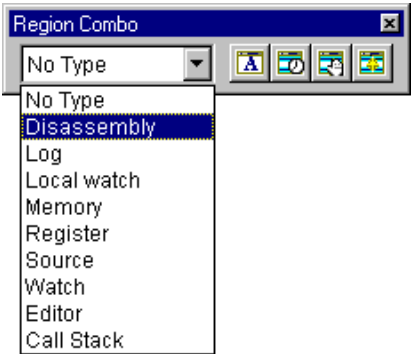


The Region toolbar lets you set or change a region's type. To stop the display from updating in all regions press CTRL+SHIFT+U.

*Table 6: 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

## 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.

Table 7: Commands on the Region Combo toolbar






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

## Splitter toolbar



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

*Table 8: 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.

*Table 9: 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
Find In Files		none available



## 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.

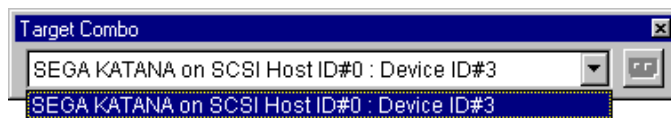
When you create a new window, CodeScape uses the target information from the selected processor on the target. The Target window provides point and click access for selecting a target processor.

---

**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.

*Table 10: 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



Table 11: Commands on the Edit toolbar

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

## 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.*


---

## 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, 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

- 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.

### 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.

### 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


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

## **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.

*Table 12: 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.

## Target window

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

*Table 13: Controlling target processor execution*

Use:	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 step commands, or run the step commands.
Breakpoints	Add, enable, disable, configure, reset, or remove data breakpoints.
Overlay Management...	View and manage file overlay properties.
Debug Info Mapping...	Set the start and end address of a base address range that you can map one or more associated address ranges from. If you do this, any debug information describing the base address range applies to the associated mapped address ranges instead.
Reset Target	Do 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.
Unload Program File Info	Clear all debug information and close the program file without removing the program file from target.
Restart	Restart the currently active program file. 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.
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 in the Target toolbar, or click .*
- *To hide the processor(s) for a target, double-click on the target in the Target toolbar, or click .*

## Input/Output window

The Input / Output window can be docked at the top and bottom of the main window, or left free floating. The Input / Output window 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. Note that text strings longer than 132 characters are truncated. 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.
- **Find In Files tab** with a list of all files containing a search string at your specified location. Double-click an item in the list to examine the associated file. To search for a string in multiple files:

### **Notes:**

1. *If you enable high level optimization when you build your project the compiler output can make source-level tracing confusing.*
2. *The window appears automatically and shows the Build, Log, and Scripts tabs, but must be opened from the Toolbar Configuration dialog to show the Find In Files tab.*

## Shortcut menus on the Input / Output window

*Table 14: The shortcut menu on the Build tab*

Use:	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.
Stop Make	Stop the active project build.
Next Error	Move to the next error in the list.
Previous Error	Move to the previous error in the list.
Print...	Print the contents of the build tab.
Save To File...	Save the contents of the build tab to a file.
Clear	Clear the contents of the Project Build window.
Allow Docking	Toggle docking for the window on or off.
Hide	Hide the window.

*Table 15: The shortcut menu on the Log tab*

Use:	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.
Overlay Management	View and manage file overlay properties.



Use:	To:
Debug Info Mapping...	Set the start and end address of a base address range to map one or more associated address ranges from. Any debug information describing the base address range then applies to the associated mapped address ranges.
Reset Log	Clear the contents of the Log tab.
Allow Docking	Toggle docking for the window on or off.
Hide	Hide the window.

*Table 16: The shortcut menu on the Scripts tab*

Use:	To:
Run Script	Select and run a script.
Clear	Clear the contents of the Script tab.
Print...	Print the contents of the build tab.
Save To File...	Save the contents of the build tab to a file.
User Scripts	This option is gray until you add a script, then the script's name appears on the menu.
Allow Docking	Toggle docking for the window on or off.
Hide	Hide the window.

*Table 17: The shortcut menu on the Find In Files tab*

Use:	To:
Clear	Clear the contents of the Find In Files tab.
Goto Line	Open the file at the first line containing the search term.
Next	Move to the next file in the list.
Previous	Move to the previous file in the list.
Allow Docking	Toggle docking for the window on or off.
Hide	Hide the window.

## 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.

### **Notes:**

- 1. Place the mouse pointer over a variable or expression to quickly view its' value.*
- 2. 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).*
- 3. If no debug information appears in a Source region, compile all source files for your project with debugging turned on.*
- 4. Shared memory address information is only available in a Disassembly region and is used when looking up file, line, and symbol information.*

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.


## 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.

*Table 18: Configuring the Source view*

Use:	To show the:
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.



*Table 19: Accessing the debugging commands in a Source region*

Use:	To click commands 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.

Table 20: Setting the cursor and the display in a Source region

Use:	To:
Set Cursor to PC	Show 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 Source and Disassembly regions

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.*

---

### 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.*


---

## 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.

## 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.*

---

## 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.

*Table 21: Configuring the disassembly view*

Use:	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.

*Table 22: Accessing the debugging commands in a Disassembly region*

Use:	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.
Overlay Management	View and manage overlay properties.

Table 23: Setting the cursor and the display in a Disassembly region

Use:	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.

### 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.*

---



### 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.

## 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 at a breakpoint, 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.

### Notes:

1. 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.
2. Use Run to Cursor to return to a specific function outside of the active one.

Table 24: The shortcut menu in a Call Stack region

Use:	To click commands to:
Show Parameter Names	Toggle 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/ Stack	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.
Auto Radix	Tell CodeScape to automatically use the appropriate radix for viewing each item.
Execution	Run, stop, and restart your program. Run a program until it executes a specified address. Run all program files simultaneously. Stop all program files 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.
Overlay Management	View and manage overlay properties.
Properties	Configure fonts and colors and set the region and processor update rates.

## 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.

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.

## 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.

*Table 25: The shortcut menu in a Watch region*

Use:	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.
Auto Radix	Tell CodeScape to automatically use the appropriate radix for viewing each item.
Edit Watch Value...	Modify the value of a variable or watch expression.

Use:	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.
Overlay Management	View and manage overlay properties.
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 a Watch region

### Add a symbol or variable

- Right-click, click Insert to add a symbol or variable at the cursor position.
- OR-
- Right-click, click Append to add symbol or a variable at the end of the 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.

### **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.

---

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

---



## 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.*

---

*Table 26: The shortcut menu in a Local Watch region*

Use:	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.
Auto Radix	Tell CodeScape to automatically use the appropriate radix for viewing each item.
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.

Use:	To:
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.
Overlay Management	View and manage overlay properties.
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.

## 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.

## Memory region

Use the Memory region to view the targets memory contents from a specific address. 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.

*Table 27: The shortcut menu in a Memory region*


Use:	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 as Hexadecimal	Display memory as hexadecimal values
Display as Unsigned Decimal	Show decimal numbers without indicating if they are positive or negative.
Display as Signed Decimal	Show decimal numbers and indicate if they are positive or negative.
Display as Floats	Display memory as floats.
Display as Doubles	Display memory as doubles.
Decimal Format	Only show the sign for negative numbers (and not positive numbers) when the display is set to Signed Decimal. Show leading zeros when the display is set to Unsigned Decimal or Signed Decimal.
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.

Use:	To click commands to:
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 step, or run the step commands.
Breakpoints	Toggle a breakpoint on or off. Enable, disable, configure, reset, and remove breakpoints.
Overlay Management	View and manage overlay properties.
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.

## Editing memory

### Change values in the Memory region

To change a byte, word, or long value in a Memory region:

- Use the + and - keys to increment or decrement the current value.  
-OR-
- Double-click or press ENTER, then type over the existing value.  
-OR-
- Right-click, click Edit memory value...  
-OR-
- Press CTRL+ALT+E.  
The Expression Evaluator dialog box appears. Enter a valid expression then click OK.

---

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

---

### Filling memory with 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 skips over any sensitive areas such as invalid memory areas, write-only memory, and memory reserved for the monitors.*

---

## 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.

Table 28: 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.

### Examples

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.

## 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,FFFFFFFF?,???????7

## 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.

*Table 29: The shortcut menu in a Register region*

Use:	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.
Show Contents at SEA/DEA	Toggle the display of values of the SEA (Source Effective Address) and the 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.
Show Float Registers As Hexadecimal	Toggle the floating point register display between decimal and hexadecimal.
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.
Overlay Management	View and manage overlay properties.
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.

## 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

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

Display registers in four columns

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

Display registers in the available area

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

## 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 appear 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.
3. 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 for a specific region. If you edit another Register region, the changes appear in the write protected region.*

---

### Enter an expression for the instruction at the 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).

### **Notes:**

1. *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.*
2. *It is recommended that the floating-point enable bits are not used when program performance is needed.*
3. *For more information about SH4 floating-point exceptions, refer to the Hitachi SH7091 Programmer's Manual.*

## Hitachi target processor register display

### General registers

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

R0 acts as a fixed source 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 appear 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.*

---

*Table 30: Flags in the Registers region after a target 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.

## 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.*

---

## 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 editing a UNIX target program file, 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).*

---

*Table 31: The shortcut menu in an Edit region*

Use:	To click commands to:
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.

Use:	To click commands to:
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.
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.

# *Interacting with target processors*

---

The File menu commands let you: create, open, close, and save sessions. You can also reset target processors, load and add program files to a project, close and restart program files, and specify large blocks of data to save from memory or to move into memory.

---

*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 reload a program file at any time, including any debug information, use Restart.*

---

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. A Hard Reset reloads an open program file's executable to the target.

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, 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 the program file's path and name.
5. In the Load Options text box, select one of the following: Load Binary Only, Load Symbols/Debug only, or Load Both Binary and Symbols/Debug.
6. Select any of the following options you require, then click OK:
  - 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.

---

**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

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.*

---

### 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.



## Restart a program

Restart reloads the open program file, including the debug information, 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 .
- -OR-
- Right-click in any region, click Execution, then click Restart.

## Configure the network settings

CodeScape provides beta support for ethernet debugging with a DASH4 debug pod. If you are using CodeScape and the DASH4 with the SH4 target hardware for the first time, you must configure the target to load the debug stub. For more information refer to the DASH3/DASH4 Getting Started Guide.



# *Working with sessions*

---

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.

**Notes:**

1. *The commands for working with sessions are on the File menu.*
2. *When you next open the session CodeScape will automatically load this information.*
3. *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.*
4. *You cannot add or remove targets during a session.*

## File menu commands

To open a new session:

1. Click File, Session New...  
The New Session 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 a project:

1. Click Project, then click Setup Project.
2. Enter project's name, for example, makefile, in one of the following ways:
  - Type the project's name and path location in the Project File text box.
  - OR-
  - Select a recent project from the Project File list.
  - OR-
  - Click Browse, then search for a project file.
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.

### 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 project

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.

---

**NOTE:**        *The Input/Output window 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 Tools, Options...and select 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.

**Notes:**

1. *If you enable high level optimization when you build your project the compiler output can make source-level tracing confusing.*
2. *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).*
3. *The toolbars provide access to the main debugging functions. Use the Toolbar Configuration check box to show or hide toolbars.*

## Loading the debug stub

You can select options to specify how the debug stub loads onto your target processor depending on your development requirements.

You can select the debug stub (ASE or Full) you want to load, and how it loads onto your target processor. To use all of CodeScape's debugging facilities you must load the Full stub. For more information refer to the Hardware Reference manual.

The DASH start-up configuration options are for:

- Telling the DASH to allow debug support and get debug data.
- Using a boot-ROM to initialize the target before loading the Extended debug stub.
- Loading the Extended debug stub from the DASH on request from the boot-ROM.
- Initializing the target and RAM without a boot-ROM.
- For loading and running the Extended debug stub at a particular address in the target's RAM without a BIOS call from the boot-ROM.
- Setting what happens after the Extended debug stub is loaded.
- Configuring target reset options.

### Loading the debug stub

1. Click Tools, Configure Target/Communication... and select Debug Support Enabled to allow debug support and get debug data.  
Reset the target and the DASH loads the 1K ASE Stub into the ASE RAM in the target processor. The ASE Stub then enables basic target debugging and does not need to use external memory. The ASE Stub is required to load the Full Stub and can be used to examine any location within the target to aid hardware debugging. As the ASE Stub allows hardware breakpoints, it is also used to debug a Boot ROM (if fitted). If the option is unchecked, after a reset, the target runs without the intervention of the DASH or any debugging facilities.
2. Then select any further options you require.
  - Select *Run Boot ROM* if you have a boot ROM which specifies the load address for the stub.  
The DASH loads the Full debug stub to the address set by the boot ROM and CodeScape is ready to start debugging.



- Select *Initialize RAM with Script* if you do not have a boot ROM and you want to initialize the target's RAM with a script. Browse for the script you want to run. An example script called examboot.js is provided on the CodeScape installation CD.

If you do not initialize the RAM at this stage, CodeScape will start with only the ASE debug stub loaded in the target processor and you will get limited debug data.

- Select *Force Load Full Stub* and specify address in the target's RAM to load the full debug stub to. The address must be on a long word boundary, that is, the two least significant bits must be zero.

If you do not check this option, or you do not set a load address for the Full debug stub, CodeScape starts with only the ASE debug stub loaded in the target processor. This also occurs if the boot ROM times out after 10 seconds.

- Select *Halt After Target Reset* to stop your application code at start up.
3. Select the Reset Method to specify how the DASH generates a target reset. Pin Reset is preferable to UDI Reset as it enables the entire target hardware to be reset. Refer to the DASH Hardware Reference Manual for pin details.
    - Select Pin Reset to tell the DASH to reset the target using its DBG\_RST# line. For this to operate the target system must have a functional DBG\_RST# and RESETIN# line available to the DASH (the DASH uses the RESETIN# line as confirmation that the DBG\_RST# line assertion occurred correctly).
- OR-
- Select UDI Reset if there is no hard-wired reset mechanism on the target hardware. With this selected, upon a hard reset, the DASH sends a serial command on the UDI TDO line to reset the target via JTAG.

You must either power cycle the DASH, or do a Hard Reset in CodeScape for the options to take effect.

#### **Notes:**

1. *The DA Start-up configuration is stored in EEPROM. When debug support is disabled (Debug Support Enabled tick box not checked) and the DASH is power cycled, the DASH would normally not be able to see the target. For this reason, when powering up, the DASH always resets debug support to enabled. If you want to run CodeScape without debug support, uncheck Debug Support Enabled and do a target Hard Reset from within CodeScape.*
2. *The version of the Full Stub to be loaded will be of the correct endian for the configuration of the target processor. This selection is made automatically by the DASH.*


## Running and stopping programs

The run options are:

- Run all processors simultaneously, and stop all processors simultaneously.
- Run a program, run a program until it executes a specified address, and 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 .
- OR-
- 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

- Click Debug, point to Execution 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

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

## Undo a step

- Click Debug, point to Execution 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, point to Execution then click Step Run In (SHIFT+F7).
- OR-
- Right-click in a region, point to 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, point to Execution then click Step Run Out (SHIFT+F8).
- OR-
- Right-click in a region, point to 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, point to Execution 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 reloads the open program file, including the debug information, and resets the PC to the entry point (if known). If the program file's last modification time has changed, symbolic information is loaded.

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

```
[TARGET DEVELOPMENT MACHINE 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 include:

- Adding breakpoints, and adding a breakpoint at the current cursor position.
- Removing breakpoints, and removing all breakpoints.
- Enabling and disabling breakpoints, and resetting breakpoints.

### 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.

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.


### 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 changes 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 changes from  to  to show that the breakpoint is disabled.


### Enable all breakpoints

- 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

- 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

- 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.

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.  
The Code Settings tab appears.
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.

## 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.

If you put a watch (data) breakpoint on a member of a union it triggers 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.

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.  
The Watch Settings tab appears.
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, session restore, or if attributes are not validated during breakpoint configuration. Disabled breakpoints do not affect code execution or use 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 a Watch Access breakpoint into a Watch Data breakpoint that uses the features of the UBC (User Break Controller). Enter the required Data Expression and click Define. The Breakpoint watch data expression dialog 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. Enter the Access Size required (the default is Any). When you use Toggle to add a watch breakpoint its size, if known, is used instead of Any.
7. Select the Access Type required. The default is Both read and write access.

## 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 start to tell the Simulator to Start when the breakpoint has been triggered.
- Select Cause processor profiling 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.  
If there is no Log region for the Target Processor, CodeScape creates one.
- Run Script and specify a script to execute when the breakpoint conditions have been met.

---

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

---

## Using the Advanced tab

---

**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.

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

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 watch breakpoint options***

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.

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 \$ 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.

*Table 32: Flags and their meanings*

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.
#	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. 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. 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. Ignored when used with c, d, i, u, or s.

## 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.

*Table 33: 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.

Character	Type	Output 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.
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.



# *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.

## 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.

*Table 34: 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.
Expression Format radio 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.

*Table 35: 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.

*Table 36: 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

Table 37: 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.

**Operator limitations:**

- Scope operator, '::'. The scope operator is valid as part of a class element name, for example, c\_basic::print.
- Assignment operators, like =, +=, \*=, ++, --, are not implemented in this release.
- File/line number format is not implemented in this release.

## Assembler expressions

*Table 38: 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
<=	Binary	expr <= expr	Less Than or Equals
>	Binary	expr > expr	Greater Than

Operator	Type	Usage	Description
>=	Binary	expr >= expr	Greater Than or Equals

Table 39: 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 'S' 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		



# 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

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.*

---

*Table 40: The shortcut menu on the Scripts tab*

Use:	To:
Run Script	Select and run a script.
Clear	Clear the contents of the Script tab.

Use:	To:
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. To assign a keyboard shortcut to the script click Tools, select Customize then click Keyboard...

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.  
Select a command in the Menu Contents box, then Use Move Up and Move Down to set where it appears on the Tools menu.
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.

## 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

#### **Description**

Loads the specified program file.

#### **Syntax**

```
LoadProgramFile(path and filename)
```

#### **Remarks**

Uses the file path as a parameter and returns 1 if a file is loaded, otherwise it returns 0.

#### VBScript example

```
LoadProgramFile( "e:\projects\maketest\hello.elf" )
```

#### JScript example

```
LoadProgramFile( "e:\projects\maketest\hello.elf" )
```

### HardReset

#### **Description**

Resets the target processor with a hard reset.

#### **Syntax**

```
HardReset()
```

#### VBScript example

```
HardReset()
```

#### JScript example

```
HardReset();
```

### SoftReset

#### **Description**

Resets the target processor with a soft reset.

#### **Syntax**

```
SoftReset()
```

#### VBScript example

```
SoftReset()
```

#### JScript example



```
SoftReset();
```

## Run

### *Description*

Runs the target processor.

### *Syntax*

```
Run()
```

VBScript example

```
Run()
```

JScript example

```
Run();
```

## WriteMessage

### *Description*

Writes a message string to the script window.

### *Syntax*

```
WriteMessage(string Message)
```

VBScript example

```
WriteMessage( "Script complete. Removing all breakpoints." )
```

JScript example

```
WriteMessage( "Script complete. Removing all breakpoints." );
```

## WriteRegister

### *Description*

Sets the specified register to the given value.

### *Syntax*

```
WriteRegister(Register value, Numeric value)
```

VBScript example

```
WriteRegister "fr0", 3.14159
```

JScript example

```
WriteRegister( "fr0", 3.14159 );
```

## RegisterValue ReadRegister

### Description

Gets the value held in the specified register.

### Syntax

```
RegisterValue ReadRegister(RegisterName)
```

### VBScript example

```
Sub ReadSomeRegisters()  
WriteMessage( "Value of pc = " & ReadRegister( "pc" ) )  
WriteMessage( "Value of r0 = " & ReadRegister( "r0" ) )  
End Sub
```

### JScript example

```
function ReadSomeRegisters()  
{  
WriteMessage( "Value of pc = " + ReadRegister( "pc" ) )  
WriteMessage( "Value of r0 = " + ReadRegister( "r0" ) )  
}
```

## LoadBinaryFile

### Description

Loads a binary file from the specified location.

### Syntax

```
LoadBinaryFile(Path and filename,Numeric binary location)
```

### VBScript example

```
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
```

### JScript example

```
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" );  
}
```

## SetBreakpoint

### *Description*

Sets a code breakpoint at the specified address.

### *Syntax*

```
SetBreakpoint(Numeric address)
```

### VBScript example

```
SetBreakpoint( "add_fn" )
```

### JScript example

```
SetBreakpoint( "add_fn" );
```

## ClearAllBreakpoints

### *Description*

Clears all breakpoints.

### *Syntax*

```
ClearAllBreakpoints()
```

### VBScript example

```
ClearAllBreakpoints()
```

### JScript example

```
ClearAllBreakpoints();
```

## RemoveBreakpoint

### *Description*

Removes the breakpoint from the specified address.

### *Syntax*

```
RemoveBreakpoint(Numeric address)
```

### VBScript example

```
RemoveBreakpoint()
```

### JScript example

```
RemoveBreakpoint();
```

## CreateBreakpoint

### Description

Creates a breakpoint of the given type at the address. Returns a breakpoint identifier on success, otherwise 0.

### Syntax

```
CreateBreakpoint(Type,Address)
```

### Remarks

To create a:

- Code Breakpoint set the Type to 0.
- Watch Breakpoint set the Type to 1.
- Simulator or Start Breakpoint set the Type to 2.
- Profiler start Breakpoint set the Type to 3.
- Profiler stop Breakpoint set the Type to 4.

In the examples, the breakpoint types are represented by the following variables:

```
BPTYPE_CODE=0;  
BPTYPE_WATCH=1;  
BPTYPE_SIMSTART=2;  
BPTYPE_PROFSTART=3;  
BPTYPE_PROFSTOP=4;
```

### VBScript example

```
Sub CreateWatchBP()  
breakID= CreateBreakpoint( BPTYPE_WATCH, "main" )  
SetWatchBreakpointParameters breakID, true,"14", BPEXPR_C,  
BPACCESSSIZE_BYTE, BPACCESSTYPE_WRITE  
End Sub
```

### JScript example

```
function CreateWatchBP()  
{  
breakID= CreateBreakpoint( BPTYPE_WATCH, "main" );  
SetWatchBreakpointParameters( breakID, true,"14", BPEXPR_C,  
BPACCESSSIZE_BYTE, BPACCESSTYPE_WRITE );  
}
```

## EnableBreakpoint

### *Description*

Enables or disables the specified breakpoint.

### *Syntax*

```
EnableBreakpoint(identifier,boolean enable)
```

### Remarks

The identifier specifies the breakpoint. Set the boolean enable value to:

- 1 to enable the specified breakpoint.
- 0 to disable the specified breakpoint.

### VBScript example

```
EnableBreakpoint breakID, true  
EnableBreakpoint breakID, false  
EnableBreakpoint breakID, false  
EnableBreakpoint breakID, true
```

### JScript example

```
EnableBreakpoint( breakID, true );  
EnableBreakpoint( breakID, false );  
EnableBreakpoint( breakID, false );  
EnableBreakpoint( breakID, true );
```

## SetBreakpointActions

### Description

Enables or disables the specified breakpoint action.

### Syntax

```
SetBreakpointActions(identifier,numeric action,boolean enable)
```

### Remarks

The identifier specifies the breakpoint. Set the boolean enable value to:

- 1 to enable the specified breakpoint action.
- 0 to disable the specified breakpoint action.

Set the numeric action value to:

- 0 to halt the breakpoint when it is hit.
- 1 to remove the breakpoint after it has been hit.
- 2 to display a message box prompt when the breakpoint has been hit.
- 3 to beep when the breakpoint has been hit.

### VBScript example

```
SetBreakpointAction breakID, BP_ACTION_HALT, true  
SetBreakpointAction breakID, BP_ACTION_ONESHOT, false  
SetBreakpointAction breakID, BP_ACTION_PROMPT, false  
SetBreakpointAction breakID, BP_ACTION_BEEP, true
```

### JScript example

```
SetBreakpointAction( breakID, BP_ACTION_HALT, true );  
SetBreakpointAction( breakID, BP_ACTION_ONESHOT, false );  
SetBreakpointAction( breakID, BP_ACTION_PROMPT, false );  
SetBreakpointAction( breakID, BP_ACTION_BEEP, true );
```

## SetBreakpointLog

### *Description*

Sets a log expression for the breakpoint specified by the breakpoint identifier.

### *Syntax*

```
SetBreakpointLog(breakpoint identifier,string expression,boolean logType)
```

### Remarks

- The identifier specifies the breakpoint.
- The expression specifies the log expression.
- Set the logType to false to always log or true to log when conditions match.

### VBScript example

```
SetBreakpointLog breakID, "Hello John", BPEXPR_C
```

### JScript example

```
SetBreakpointLog( breakID, "Hello John", BPEXPR_C );
```

## SetBreakpointScript

### *Description*

Attaches a script to a breakpoint.

### *Syntax*

```
SetBreakpointScript(identifer,string script path,numericscripttype, string script arguments,boolean prompt)
```

### Remarks

- The identifier specifies the breakpoint.
- The script path is the file path for the script. Set the script type: 0 for JScript, or 1 for VBScript
- The script arguments is a string holding the script's arguments. Set the prompt to 1 to request arguments when the breakpoint triggers, or to 0 otherwise.

### VBScript example

```
SetBreakpointScript breakID,  
"e:\\projects\\codescape\\debugs\\testscript.js", BPScript_JSCRIPT,  
"arg1 arg2 arg3", false
```

### JScript example

```
SetBreakpointScript( breakID,  
"e:\\projects\\codescape\\debugs\\testscript.js",  
BPSCRIPT_JSCRIPT, "arg1 arg2 arg3", false );
```

## SetBreakpointCondition

### Description

Sets a conditional expression for the breakpoint.

### Syntax

```
SetBreakpointCondition(identifier,string expression,numeric  
expression type,numeric trigger count,boolean incOnTrue,boolean  
breakWhen)
```

### Remarks

- The identifier specifies the breakpoint. The expression is a string representing the condition.
- Set the expression type to 0 for C/C++, or to non-zero for assembly.
- The trigger count is the number of hits before breakpoint actions are performed.
- Set incOnTrue increment the trigger count only when conditions are true, or set incOnFalse to always increment the trigger count.
- Set breakWhen as false to break either when the trigger reaches 0 or when condition is true. Set breakWhen as true to break when trigger reaches zero and the condition is true.

### VBScript example

```
setBreakpointCondition breakID, "index == 375", BPEXPR_C, 37, true,  
true
```

### JScript example

```
setBreakpointCondition( breakID, "index == 375", BPEXPR_C, 37, true,  
true );
```

## BOOL SetWatchBreakpointParameters

### Description

Sets the parameters for a watch breakpoint.

### Syntax

```
BOOL SetWatchBreakpointParameters(Identifier,Boolean  
incDataCondition,string dataCondition,numeric  
expressionType,numeric accessSize,numeric accessType)
```

### Remarks



- The identifier specifies the breakpoint.
- IncDataCondition is a boolean string to include a data condition.
- DataCondition is a numeric expression that sets the data condition.
- expressionType: a numeric expression that sets the expression type.
- accessSize: a numeric expression that sets the access size. To set the accessSize to:
  - Any enter the value 0.
  - Byte enter the value 1.
  - Word enter the value 2
  - Long enter the value 4.
  - Quad enter the value 8.
- accessType: is a numeric expression that sets the type of access (read, write, or both). To set the accessType to:
  - Read enter the value 1.
  - Write enter the value 2.
  - Read or Write enter the value 3.

#### VBScript example

```
SetWatchBreakpointParameters breakID, true,"14", BPEXPR_C,  
BPACCESSSIZE_BYTE, BPACCESSTYPE_WRITE
```

#### JScript example

```
SetWatchBreakpointParameters( breakID, true,"14", BPEXPR_C,  
BPACCESSSIZE_BYTE, BPACCESSTYPE_WRITE );
```

### SetBreakpointLocationMask

#### *Description*

Select a location mask for the breakpoint. The identifier specifies the breakpoint, and maskSelect is a value that specifies the bits to mask.

To mask:

- No bits, enter the value 1.
- The lower 10 bits enter the value 2.
- The lower 12 bits, enter the value 3.
- The lower 16 bits, enter the value 4.
- The lower 20 bits, enter the value 5.
- All bits, enter the value 6.

#### *Syntax*

```
SetBreakpointLocationMask(breakID,maskSelect)
```

#### VBScript example

```
SetBreakpointLocationMask breakID, BPLOCMASK_LOW10
```

#### JScript example

```
SetBreakpointLocationMask( breakID, BPLOCMASK_LOW10 );
```

### SetBreakpointDataMask

#### Description

Sets the data mask for a watch breakpoint. The identifier specifies the breakpoint, and the data mask must be a 32-bit number.

#### Syntax

```
SetBreakpointDataMask(identifier,data mask)
```

#### VBScript example to mask 10 bits

```
SetBreakpointDataMask breakID, 0x000003FF
```

#### JScript example

```
SetBreakpointDataMask( breakID, 0x000003FF );
```

### ReadByte

#### Description

Reads a byte from the specified area of memory.

#### Syntax

```
ReadByte(Numeric address)
```

#### VBScript example

```
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
```

#### JScript example

```
function ReadSomeMemory()  
{  
WriteMessage( "Byte at main = " + ReadByte( "main" ) );  
WriteMessage( "Word at main + 4 = " + ReadWord( "main + 4" ) );  
WriteMessage( "Long at main + 8 = " + ReadLong( "main + 8" ) );  
}
```

## ReadWord

### *Description*

Reads a word from the specified area of memory.

### *Syntax*

**ReadWord**(Numeric address)

### VBScript example

```
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
```

### JScript example

```
function ReadSomeMemory()  
{  
WriteMessage( "Byte at main = " + ReadByte( "main" ) );  
WriteMessage( "Word at main + 4 = " + ReadWord( "main + 4" ) );  
WriteMessage( "Long at main + 8 = " + ReadLong( "main + 8" ) );  
}
```

## ReadLong

### *Description*

Reads a long from the specified area of memory.

### *Syntax*

**ReadLong**(Numeric address)

### VBScript example

```
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
```

### JScript example

```
function ReadSomeMemory()  
{  
WriteMessage( "Byte at main = " + ReadByte( "main" ) );  
WriteMessage( "Word at main + 4 = " + ReadWord( "main + 4" ) );  
WriteMessage( "Long at main + 8 = " + ReadLong( "main + 8" ) );  
}
```

## WriteByte

### Description

Writes a byte from the specified area of memory.

#### Syntax

**WriteByte**(Numeric address, Numeric value)

### VBScript example

```
Sub WriteSomeMemory()  
WriteByte "main", 255  
WriteWord "main + 4", "0xabcd"  
WriteLong "main + 8", "0xfedcba"  
End Sub
```

### JScript example

```
function WriteSomeMemory()  
{  
WriteByte( "main", 255 );  
WriteWord( "main + 4", "0xabcd" );  
WriteLong( "main + 8", "0xfedcba" );  
}
```

## WriteWord

### Description

Writes a word from the specified area of memory.

#### Syntax

**WriteWord**(Numeric address, Numeric value)

### VBScript example

```
Sub WriteSomeMemory()  
WriteByte "main", 255  
WriteWord "main + 4", "0xabcd"  
WriteLong "main + 8", "0xfedcba"  
End Sub
```

### JScript example

```
function WriteSomeMemory()  
{  
WriteByte( "main", 255 );  
WriteWord( "main + 4", "0xabcd" );  
WriteLong( "main + 8", "0xfedcba" );  
}
```

## WriteLong

### Description

Writes a long from the specified area of memory.

### Syntax

```
WriteLong(Numeric address, Numeric value)
```

### VBScript example

```
Sub WriteSomeMemory()  
WriteByte "main", 255  
WriteWord "main + 4", "0xabcd"  
WriteLong "main + 8", "0xfedcba"  
End Sub
```

### JScript example

```
function WriteSomeMemory()  
{  
WriteByte( "main", 255 );  
WriteWord( "main + 4", "0xabcd" );  
WriteLong( "main + 8", "0xfedcba" );  
}
```

## GetParam

### Description

Returns a specific parameter.

### Syntax

```
GetParam(short param)
```

### VBScript example

```
Sub DisplayParameters()  
NumParams=GetParamCount  
WriteMessage( "Number of parameters = " & NumParams )  
For i = 1 To NumParams  
WriteMessage( "Parameter " & i & " = " & GetParam( i - 1 ) )  
Next  
End Sub
```

### JScript example

```
function DisplayParameters()  
{  
NumParams=GetParamCount()  
WriteMessage( "Number of parameters = " + NumParams );  
for( i = 0; i < NumParams; i++ )  
{  
WriteMessage( "Parameter " + i + " = " + GetParam( i ) )  
}  
}
```

## GetParamCount

### **Description**

Returns the number of parameters passed to the script.

### **Syntax**

```
GetParamCount()
```

### VBScript example

```
Sub DisplayParameters()  
NumParams=GetParamCount  
WriteMessage( "Number of parameters = " & NumParams )  
For i = 1 To NumParams  
    WriteMessage( "Parameter " & i & " = " & GetParam( i - 1 ) )  
Next  
End Sub
```

### JScript example

```
function DisplayParameters()  
{  
    NumParams=GetParamCount()  
    WriteMessage( "Number of parameters = " + NumParams );  
    for( i = 0; i < NumParams; i++ )  
    {  
        WriteMessage( "Parameter " + i + " = " + GetParam( i ) )  
    }  
}
```

## IsRunning

### Description

Returns 1 if running, 0 if not running.

### Syntax

**IsRunning()**

### VBScript example

```
Sub DisplayParameters()  
NumParams=GetParamCount  
WriteMessage( "Number of parameters = " & NumParams )  
For i = 1 To NumParams  
    WriteMessage( "Parameter " & i & " = " & GetParam( i - 1 ) )  
Next  
End Sub
```

### JScript example

```
function DisplayParameters()  
{  
    NumParams=GetParamCount()  
    WriteMessage( "Number of parameters = " + NumParams );  
    for( i = 0; i < NumParams; i++ )  
    {  
        WriteMessage( "Parameter " + i + " = " + GetParam( i ) )  
    }  
}
```

**ConfigureTraceHistory**

**Description**

Specifies the events saved in the Trace history.

**Syntax**

```
ConfigureTraceHistory(numeric Setting,boolean Enable)
```

Remarks

Events	Setting
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

**VBScript example**

```
ConfigureTraceHistory TH_LOGEXCEPT + TH_LOGSUB, true
```

**JScript example**

```
ConfigureTraceHistory( TH_LOGEXCEPT + TH_LOGSUB, true );
```



## DisplayTraceHistory

### ***Description***

Displays the current history in the script's window.

### ***Syntax***

```
DisplayTraceHistory()
```

### VBScript example

```
DisplayTraceHistory()
```

### JScript example

```
DisplayTraceHistory();
```

## ClearDisplay

### ***Description***

Clear the script output window.

### ***Syntax***

```
ClearDisplay()
```

### VBScript example

```
ClearDisplay()
```

### JScript example

```
ClearDisplay();
```



# 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]...
```

*Table 41: 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.*

---

Table 42: Command-line switches

Use this switch:	To:
<code>[-/]?</code>	Run CodeScape and view Help on using the command-line.
<code>[-/]/nologo</code>	Run CodeScape without the splash screen appearing.
<code>[-/]/c</code>	View information on the Log tab as each command of the Cross Products Fileserver library (libcross) executes. (Crosslib Verbose mode.)
<code>[-/]/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>[-/]/autoload</code>	Run CodeScape using the last loaded session file.
<code>[-/]/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".

## Unloading a program file from the command-line

When you unload a program file from the command-line, use the switch format:

```
-t#p#u
```

For example, CodeScape -t1p1u unloads the program file loaded on target 1, processor 1.

Table 43: 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-.



# *Configuration file: dali.cfg*

---

Memory ranges and processor timing information are specified in the configuration file `dali.cfg`. If you cannot see or access specific areas of memory that you require, you can edit `dali.cfg` to configure CodesScape with the areas of memory you want to read. If you edit `dali.cfg`, first make a copy of the original file.

Dali.cfg has three sections:

- Valid memory.
- Shared memory.
- Software breakpoint settings.

## Valid memory

### Description

Specifies valid areas of memory for your code.

### Syntax

```
Access Method = Start Address, End Address, Size, Expression,  
Restrictions, Byte, Word, Long, Quad, Cache, Byte, Word, Long, Quad,  
Cache
```

### Remarks

The Access Method, Start Address, End Address, and Size fields must be completed and contain valid values. The Expression, Restrictions, Byte Word, Long, Quad, and Cache fields are optional.

- Access Method can be: Read, Write, or ReadWrite.
- Size can be: BYTE, WORD, or LONG.  
Size controls the size of commands sent to the target. For example, if you display a byte where the memory access size is LONG, 4 bytes will be read by the debug stub, but only the byte required will be displayed.
- The expression 'expr' column is disabled for SH4.
- The restriction "SimulatorHardwarePort" allows the simulator to access this area but not the memory region.
- The simulator timings are used to calculate execution times when memory is accessed. A value of "-1" indicates that an operation is not possible.

## Shared memory

### Description

Marks specific mirrored memory images as shared memory.

### Syntax

```
SharedMemory = Start Address, End Address, Label
```

### Remarks

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



To avoid this, mark mirrored memory images as shared memory in the configuration file dali.cfg. The example file 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.

## Example configuration file

```
[DASH MasterSH4EVA.ValidMemory]
; General Address Space - optimal transfer size is LONG
; Access
; Method Start End Size Expr Restrictions Byte Word Long Quad Cache WRITE READ
ReadWrite = 0x0C000000, 0x0CFFFFFF, LONG, , NoRestrictions , 4, 4, 4, 16, 7, 7, 7, 28
ReadWrite = 0x0D000000, 0x0DFFFFFF, LONG, , SimulatorHardwarePort, 1, 4, 4, 1, 7, 1, 1, 1
ReadWrite = 0x0E000000, 0x0EFFFFFF, LONG, , NoRestrictions , 4, 4, 4, 16, 7, 7, 7, 28
ReadWrite = 0x0F000000, 0x0FFFFFFF, LONG, , NoRestrictions , 4, 4, 4, 16, 7, 7, 7, 28
ReadWrite = 0x40000000, 0x44ffffff, LONG, , NoRestrictions , 1, 1, 1, 1, 1, 1, 1, 1
ReadWrite = 0x40000000, 0x44ffffff, LONG, , NoRestrictions , 1, 1, 1, 1, 1, 1, 1, 1
; Areas for Java Script
ReadWrite = 0x7F000000, 0x7F00001F, LONG, , NoRestrictions , 1, 1, 1, 1, 1, 1, 1, 1
ReadWrite = 0x7F800000, 0x7F800003, LONG, , NoRestrictions , 1, 1, 1, 1, 1, 1, 1, 1
ReadWrite = 0x7F800004, 0x7F800005, WORD, , NoRestrictions , 1, 1, 1, 1, 1, 1, 1, 1
ReadWrite = 0x7F800008, 0x7F800017, LONG, , NoRestrictions , 1, 1, 1, 1, 1, 1, 1, 1
ReadWrite = 0x7F80001C, 0x7F80001D, WORD, , NoRestrictions , 1, 1, 1, 1, 1, 1, 1, 1
ReadWrite = 0x7F800020, 0x7F800029, WORD, , NoRestrictions , 1, 1, 1, 1, 1, 1, 1, 1
Write = 0x7F940190, 0x7F940190, BYTE, , NoRestrictions , 1, 1, 1, 1, 1, 1, 1, 1

[DASH MasterSH4EVA.SharedMemory]
SharedMemory = 0x0C000000, 0x0CFFFFFF, Tag:B
SharedMemory = 0x8C000000, 0x8CFFFFFF, Tag:B
SharedMemory = 0xA0000000, 0xA0FFFFFF, Tag:B

[DASH MasterSH4EVA.Settings]
AutoSoftBreakEnabled = 0
```

## Software breakpoint settings

### **Description**

Specifies to either use, or not use software breakpoints during tracing.

### **Syntax**

```
AutoSoftBreakEnabled = 0
```

### **Remarks**

AutoSoftBreakEnabled controls the use of soft breakpoints (insertion of ASE-Break op-code) during tracing. Set the value to 0 (default) to disable to tell CodeScape not to use software breakpoints and 1 to enable it.

---

**CAUTION:** *Do not use software breakpoints when debugging interrupt routines. Under certain conditions the SH4 will cause a manual reset of the target.*

---

# Shortcut keys

---

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.

## File menu ALT+F

*Table 44: 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

Option	Keyboard shortcut
Hard Reset	ALT+F2
Load Program File...	CTRL+SHIFT+C
Unload Program File Info	CTRL+ALT+U
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

*Table 45: 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***Table 46: Keyboard shortcuts for the View menu commands*

Option	Keyboard shortcut
Toolbar...	none available
Status Bar	none available
Properties...	none available

**Project menu ALT+P***Table 47: 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***Table 48: Keyboard shortcuts for the Debug menu commands*

Option	Keyboard shortcut
Execution	none available
Run All	CTRL+F9
Stop All	none available
Run/Stop	F9

Option	Keyboard shortcut
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
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
Overlay Management	CTRL+ALT+O
Debug Info Mapping...	CTRL+SHIFT+M

Option	Keyboard shortcut
Set Cursor to PC	CTRL+SHIFT+P
Set PC to Cursor	CTRL+ALT+P
Goto Address...	CTRL+G

## Region menu ALT+R

*Table 49: 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

*Table 50: Keyboard shortcuts for the Tools menu commands*

Option	Keyboard shortcut
Simulate Processor	CTRL+ALT+Z
Profiler	CTRL+ALT+X
Find In Files	none available
Workshop	none available
Scripts	none available
Run Script...	none available
Configure Target/Communication...	none available
Customize	none available
Keyboard...	none available
Tools...	none available
Scripts...	none available
Options...	none available



**Window menu ALT+W***Table 51: 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***Table 52: Keyboard shortcuts for the Help menu commands*

Option	Keyboard shortcut
Help Topics...	none available
Keyboard	none available
About CodeScape...	none available

## New Windows

*Table 53: 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

## Log tab

*Table 54: Keyboard shortcuts on the Log tab*

Option	Keyboard shortcut
Configure Log...	none available
Print...	none available
Save to File...	none available
Reset Log	none available

## Miscellaneous commands options

*Table 55: Keyboard shortcuts for miscellany*

Option	Keyboard shortcut
Toggle Window Headers	CTRL+ALT+H

Option	Keyboard shortcut
Evaluate Expression	CTRL+E
Cycle Radix	CTRL+H

## Profiler options

*Table 56: 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

*Table 57: Keyboard shortcuts for the Disassembly region commands*

Option	Keyboard shortcut
Synchronize Cursor	none available
Source Annotation	CTRL+SHIFT+A
Show Profile Hits	none available
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
Display As (Source, Disassembly, or Both)	none available
Evaluate	none available
Tools	none available
Find...	CTRL+F
Find Next	F3
Disassemble to File...	none available

## Source region

*Table 58: Keyboard shortcuts for the Source region commands*

Option	Keyboard shortcut
Show Profile Hits	none available
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
Goto First Line of Source File	CTRL+HOME
Goto Last Line of Source File	CTRL+END

## Call Stack region

*Table 59: 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

Option	Keyboard shortcut
Auto Radix	none available

## Editor region

*Table 60: 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+F2
Next	F2
Previous	SHIFT+F2
Delete All	CTRL+SHIFT+F2

## Editor Keys

Table 61: Keyboard shortcuts for the Editor Keys commands

Option	Keyboard shortcut
Select	none available
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
Move To Top of File	CTRL+HOME
Move To Bottom	CTRL+END

Option	Keyboard shortcut
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
View Whitespace	CTRL+SHIFT+B

## Local Watch region

*Table 62: 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
Auto Radix	none available
Edit Local Value...	CTRL+ALT+E
Highlight Changes	none available



Option	Keyboard shortcut
Cache Expanded Symbols	none available

## Memory region

Table 63: 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 as Hexadecimal	none available
Display as Unsigned Decimal	none available
Display as Signed Decimal	none available
Display as Floats	none available
Display as Doubles	none available
Decimal Format	none available
Only show sign on negative	none available
Show leading zeros	none available
Display ASCII	CTRL+A
Display MBCS	none available
Nudge MBCS decoding	none available
Highlight Changes	none available
Set Bytes Per Line...	CTRL+SHIFT+L
Edit ASCII	CTRL+ALT+A
Edit Memory Value...	CTRL+ALT+E

Option	Keyboard shortcut
Follow Pointer	CTRL+T
Write Protect	CTRL+ALT+W
Change Inc/Dec Value	none available
Tools	none available
Find...	CTRL+F
Find Next	F3
Fill...	none available
Hex Dump to File...	none available

## Register region

*Table 64: Keyboard shortcuts for the Register region commands*

Option	Keyboard shortcut
Increment Register	NUM +
Decrement Register	NUM -
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

Option	Keyboard shortcut
Tools	none available
Save Registers	none available
Restore Registers	none available

## Watch region

*Table 65: Keyboard shortcuts for the Watch region commands*

Option	Keyboard shortcut
Delete	DELETE
Open	RIGHT ARROW
Close	LEFT ARROW
Insert	CTRL+I
Append	CTRL+A
Promote to Root	CTRL+ALT+R
Show Octal	none available
Show Decimal	none available
Show Hexadecimal	none available
Auto Radix	none available
Edit Watch Value...	CTRL+ALT+E

## Symbols

*Table 66: Keyboard shortcuts for the Symbol Completion dialog box*

Option	Keyboard shortcut
Symbol Complete	ALT+S
Choose Global/Static	ALT+G

## Build

*Table 67: Keyboard shortcuts for the Project Build commands*

Option	Keyboard shortcut
Next Error	F4
Previous Error	SHIFT+F4

## Target Settings

*Table 68: Keyboard shortcuts for configuring the target processor*

Option	Keyboard shortcut
Configure Processor	none available

## ToolBars

*Table 69: Keyboard shortcuts for displaying the toolbars*

Option	Keyboard shortcut
Breakpoint	none available
Debug	none available
Processor Combo	none available
Input / Output	none available

Option	Keyboard shortcut
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

